GLBasic Programmers Reference Guide

(Second Edition – First Revision)

A programmers guide to using GLBasic commands. Also contains 11 tutorials.

By Nicholas Kingsley

Tutorials by BigSofty and Gernot Frisch.
Updated and revised by Cliff3D and Nicholas Kingsley
Android signing section originally posted by Dabz in the GLBasic forums

Based on the contents of the GLBasic Wiki
(http://wiki.just-do-i-t.com/index.php?title=GLBasic_Wiki)
by Nicholas Kingsley, Cliff3D et al

Copyright ©Nicholas Kingsley 2011

ISBN #: 978-1-4477-2272-4

The book author retains sole copyright to his or her contributions to this book.

Designed By Nicholas Kingsley

GLBasic is copyright ©Dream Design Entertainment 2008

Revision : 4.0.0.4

For errors and omissions, I can be contacted at njk@un-map.com

Last revised on : 16/02/12

This book is acknowledged, but not endorsed by
Dream Design Entertainment (www.glbasic.com)

The presence of the Wiz, Pandora, Linux, Windows, Android, Apple and HP logos (which are trademarks of their respective companies) are only present to show what platforms GLBasic supports, and thus no company is endorsed by or for the author.

Introduction

Welcome to the GLBasic Programmers Reference guide. This weighty tome is designed to help the beginner and more experienced GLBasic user get the most out of the language.

Covering topics such as sound, music, sprites, 3D objects as well as revised tutorials this should help everyone get to grips with the language.

Tutorials have been revised from those that come with the language, which take into account the improvements that GLBasic has had since it was originally written.

The language details have been revised from the Wiki entries (at **http://wiki.just-do-i-t.com/index.php?title=GLBasic_Wiki**)

V9 of GLBasic has introduced a new platform to the fold : The Palm Pre. It has also updated the Windows input system, updated the editor and added a few new commands

PART #1

GLBasic Tutorials

Tutorial #1 (General)

Hello and welcome to GLBasic, your new programming language.

Here you will learn how to convert your ideas into real programs.

A little word of warning: GLBasic cannot do wonders, it has its strengths and weaknesses. A good programmer excels by utilising the strengths of a language and working around its weaknesses.

Having said that, there are many areas where GLBasic excels - sprites for example. You will find sprite rotations, alpha blending and zooming, while difficult in some languages, are simple to implement in GLBasic. To reach a stable frame rate, GLBasic can easily be slowed down to a certain number of frames per second to match any target platform.

A program is made up of very simple commands that individually do almost nothing. Combining these simple commands to make them do something powerful is the difficult thing about programming.

Programs can only:

- Declare variables (named containers for data)
- Make jumps to other parts of the program
- Manipulate the data held in variables
- Transfer variables to functions
-

A compiler takes these commands and converts them into a language the computer can understand.

General

GLBasic is a variant of the BASIC programming language and a very easy one at that. It has its own syntax and if you don't keep within this syntax 100% of the time, the GLBasic compiler will display an error asking you to correct your syntax before it will compile your code.

GLBasic finishes every command with a ';' (semicolon) or a new line, although this is usually optional, unless you want to put more than one command on a line.

Some examples of using a semicolon include :

```
a%=1;
test(); PRINT “Hello”,0,0; PRINT “More text”,0,100,TRUE
```

Semicolons should not be used to separate variable declarations in functions, although they can be used when defining variables in a type.

All commands in GLBasic must be written in capital letters. The editor will automatically convert the GLBasic commands you type to upper-case even if you do not. If it fails to convert a command, check that you have spelt it correctly.

Every character to the right of '//' on a line will be seen as a comment within the code. Comments will be ignored when the compiler converts your code into a program. This is handy for adding notes within your source. You can uncomment or comment several lines at once within the GLBasic IDE by selecting the lines and then pressing the Comments toolbar button:

GLBasic uses hidden surfaces (Double Buffering) for graphical display. If you code something in GLBasic you will see nothing before your program calls the `SHOWSCREEN` command.

`SHOWSCREEN` takes the current back buffer, makes it visible, then creates a new back buffer with a predefined image or a cleared image. If you don't see any result from your program, the first thing to check is whether you have forgotten to add the `SHOWSCREEN` command to it.

Think of Double Buffering as two sheets of paper, one on top of the other where you draw on the bottom sheet. Once you're finished drawing, you swap the sheets and in the process erase the new bottom sheet. This hides the actual drawing of the sheet from the user and in the process eliminates screen flicker.

Our First Program

Start the editor and select "Create a new project" in the wizard. Type "Hello World" within the "Name of Project" box and press OK.

Then type the following program into the editor's window.

```
// ------------------------------- //
// Project: Hello World
// My First Program
// ------------------------------- //
PRINT "HELLO WORLD!" ,100,100
SHOWSCREEN
MOUSEWAIT
END
```

Save all your work with the Double-disk toolbar button:

or use the menu option "File/Source code file/Save all".

Compile the program using the Sand-bucket toolbar button:

or the menu option "Compiler/Build".

Finally, start the program with either the "Start" toolbar button:

or the menu option "Compiler/Start".

For more detailed information on the editor, please refer to the *GLBasic User Guide* book by Nicholas Kingsley, which is available from http://www.lulu.com/product/paperback/glbasic-user-guide/12674927

You should see a black screen with "HELLO WORLD" printed on it. You may have trouble reading the text – this is due to the fact that there is no proper font to use. For more details, see the *2D Graphics* section below.

After clicking your mouse button the program will exit. You have now made your first step into the great new world of programming!

Enough of the simple stuff, let's get started with 'variables'!

Declaring variables. There are 3 'Types' of variables native to GLBasic (As of version 8, GLBasic can also use some C/C++ variable types – this is discussed much later on). These native variable types are Floating-point numbers, Integers and Strings.

You can think of Variables as desk-drawers. On the front of the drawer there's the name of the variable and within the drawer you'll find the value of the variable. Variables should be declared, as GLOBAL or LOCAL, before their first use, but de-activating the "Explicit Declarations" options in the Project → Options menu will allow you to use variables without the need to declare them.

For brevity, these declarations will be kept to a minimum throughout this tutorial, only being SHOWN once per data type - but variables will always have to be declared in your programs!

Let's start with numbers and see how it works.

Numeric Variable Types

Please note : With the release of version 9, any variables defined by LET need to be defined before use.

```
LOCAL a
LET a=5
```

We use the command 'LET' to tell the compiler we'd like to have the variable 'a' store the value 5 within it. To make variable declarations somewhat faster to type, the 'LET' command can be left out. The resulting line would look like this:

```
a=5
```

and

```
LET a=5
LET a=a+1
```

will look like this:

```
a=5
a=a+1
```

What are we doing here? Firstly we are taking the variable 'a' and assigning ('=') the value '5' to it. We then ask the compiler to take the variable with the name 'a' again and assign it the value ('=') of 'a+1'. As 'a' was already set to the value 5 at this point, 'a' will now be given the value of 'a+1' - i.e. '5+1', which is '6'. We can also increase a by a number using:

```
INC a,1
```

which can itself be abbreviated to

```
INC a
```

as the default amount to INCrease by is one.

There are other operations that can be done with numbers. GLBasic uses the standard computer symbols for mathematical calculations.

There are :

- '+' = Addition,
- '-' = Subtraction,
- '*' = Multiplication
- '/' = Division

with INC (increase) and DEC (decrease) being useful shorthand for simple additions and subtractions.

ATTENTION:

```
LET a= 3+4*5
```

would be processed as 3+(4*5) = 23. This has changed from older versions of GLBasic which would not evaluate it this way. GLBasic now has the following hierarchy for evaluating terms:
(brackets first) * / + - AND OR < > =
There are some special mathematical functions in GLBasic that work differently to the rest: `SIN()`, `COS()`, `TAN()` -These functions return a value.

So, for example, `a=COS(10)`, will put the mathematical result of Cosine 10 into the variable 'a'.

Another example of a numeric function is the `RND()` command. `RND()` returns a random number in the range 0 to argument.

For example:

```
LET a=RND(50) // a= random number from 0 to 50
```

When you create a variable for the first time (e.g. using LOCAL/GLOBAL command) you can append a '%' character to its name to indicate that this variable is an integer number. (An integer is a number without a fraction part).

Appending an optional '#' will make the variable a floating point number (one with a fractional part). The appendices ('%' or '#') are optional for further use of the variable.

Note : If you wish to perform integer only calculations, you can ensure this by using the function INTEGER().

String Variable Types

String variables are specified differently to numeric variables by having a '$' at the end of their name, which must be present whenever you want to use the variable.

Note : Numeric variable can swap their values with String variables and vice-versa, this process is known as a "type conversion" (i.e. converting a variable from one type - float, integer, string - to one of the others). All type conversions are handled automatically. More on this later.
String Variable Sample :

```
LOCAL a$
LET a$="HELLO"
```

We know that strings names end with '$', but how do you define the text that is to be stored within the String variable?

To allocate a string of text to a variable you surround the text with quotation marks (" ") - a$="HELLO" for example will store the text "HELLO" within the variable 'a$'.

The command LET allows you to do some useful things with strings.

```
LET a$="My"
LET b$="favourite number:"
LET c= 7
// Now look!!
LET c$=a$ + " " + b$ + " : " + c
PRINT c$,0,20
SHOWSCREEN
```

Output: My favourite number: 7

The 'LET' command, as always, can be left out if you like. You can see from this example that numbers can be converted to strings ('c$=c'), and strings can be joined together ('a$=a$+b$').

Why is this useful? Assume within a program you want to load an image from a file where the image's file-name is "image5.dat". The following is an example of how the automatic type conversion can make programming easier for the coder.

```
LET imagenumber=5 // This happens somewhere in the program...
LET image$="image" + imagenumber + ".dat"
```

Yes, you could just write `LET image$="image5.dat"` in this basic example, but as you learn GLBasic you will see where this sort of thing can come in really handy.

GLBasic offers some special functions for handling Strings.

```
INPUT name$, x, y
```

This command will cause a blinking cursor to appear at the screen position (x, y) and wait for the user to input something on the keyboard. After the user hits the return key, the variable 'name$' will contain the string of characters that was just typed. This works with numbers too (i.e. `INPUT number, x,y`).

```
dest$=MID$(source$, start, length)
```

You can use this function to obtain any part of a string. The string **dest$** will contain the resulting text from the string **source$** starting at the character indicated by **start** and running for the number of characters indicated by **length**. The location (index) of the first letter is 0, not 1.

Example :

```
name$="My house is blue"
m$=MID$(name$, 3, 5)
PRINT m$ , 20, 20
SHOWSCREEN
```

Output: house

Say you had a person typing their name in for a high score list but only wanted the first 7 letters. To achieve this you can trim the names by simply copying just the first 7 letters of the original string into a sub-string.

```
INPUT enter$, 100, 100
name$ = MID$(enter$, 0, 7)
PRINT name$, 100, 100; SHOWSCREEN; MOUSEWAIT
```

The `FORMAT$` command can convert numbers to strings more efficiently. Keep this in mind as you might need it sometime.

```
fmt$=FORMAT$(numLetters, numAfterComma, number)
```

Data Arrays Types

Now you have an overview of variables, but how would you store and manipulate large blocks of numbers or text? How would you store 1000 names in your program for example? 1000 separate variables? You could do this but it is not a practical solution - for this you would use something called

an "array".

An array can be imagined as a checker-board At each location on the checker-board is a value. To create an array in GLBasic that is 8 fields in its x-dimension and 8 fields in its y-dimension, you would define it like this:

```
LOCAL checker[] // we do not specify how many dimensions the array has when declaring the variable
DIM checker[8][8]
```

The computer will allocate memory (DIM) to store an 8x8 array, set each field to 0 and allow you to reference it via the name 'checker'.

If we want the field in location 3 across, 4 down to have a value of 7, we would write this:

```
LET checker[2][3]=7
```

Why "[2][3]" and not "[3][4]"? The array is indexed starting at 0 rather than one - i.e. the fields are numbered from 0 to 7 across and 0 to 7 down. 0,0 = top left of array, 7,7 = bottom right of array - just like map references. This is somewhat confusing at the beginning, but you'll get used to it very soon.

You can also reference an array's field via variables:

```
DIM checker[8][8]
LET checker[2][3]=7
LET x=2
LET y=3
LET a=checker[x][y]
PRINT "Checkerboard[2][3] has the value:", 100, 80
PRINT a, 100, 100
```

Output:

Checkerboard[2][3] has the value: 7

Note : Arrays may not have more than 4 dimensions.

```
DIM a[1][2][3][4]; // just OK
Data arrays can contain Strings too. Samples:
DIM space$[10][10][10] // A space with x, y and z (10 checkerboards each)
// each 10 fields (0-9)
LET space$[5][7][3]="Green"

DIM name$[5] // 5 strings
LET name$[0]="Peggy"
LET name$[1]="Tim"

DIM cinemavisitor$[30][10]
LET cinemavisitor$[seat][row]=" Tom"
```

With the command REDIM you can change the dimensions of existing arrays but keep the data held within. If you resize the array smaller than it was originally, the data outside the new array is lost.

```
DIM name$[5] // 5 strings
LET name$[0]="Peggy"
LET name$[4]="Tim"
```

```
REDIM name$[2] // 2 Words
```

The value at position 4 ("Tim") would be lost when the REDIM command is executed.
If you want to copy an array :

```
LOCAL a[]
LOCAL b[]
DIM a[5]
b[] = a[]
```

This re-dimensions a new array b[5] and copies the data from a[] to it. Note that it does not CREATE the array = as with a[], you must have previously declared the b[] array.

You can also remove rows from an array with the DIMDEL command. This is very convenient for managing a dynamic number of elements.

The last value of an array (I'll use an array called "a" in this example) is `a[ LEN(a[])-1 ]` or simply `a[-1]`. The value prior to the last value is `a[-2]`.
You can easily clear the contents of an array with the command REDIM *arrayname*[0].

Jump-marks

Problem: The program runs once and then exits but we want it to keep running.
Solution: We put a command in the program (in this case, at the end) to make it jump back to an earlier point within the program (in this case the start).
Example :

```
// A GOTO sample
beginning:
LET a=RND(600)
LET b=RND(400)
PRINT "Wow",a ,b
SHOWSCREEN
MOUSEWAIT
GOTO beginning
```

What is this doing? Firstly it defines a jump-mark with the name 'beginning' (indicated by the ":" at the end of the word). The program prints "Wow" to a random position and displays it on the screen until a mouse button is pressed. The GOTO command at the end makes the program jump to the line labelled 'beginning'. The example here shows what is called an "infinite loop" - i.e. the program will never end. To break the program out of the loop you push the "ESC" key. Keep this in mind - pushing ESC will exit your program whenever you want to.

Program jumps can also be used to write 'blocks' of code called subroutines. These are a great way of organising and debugging your code. To define a subroutine block use the menu command Project → New SUB, then use the wizard to create a new subroutine block (SUB) within your project. The GOSUB command can be used at any time to call your new subroutine block (SUB) of code after which time it will continue where it left off. An example:

```
// A GOSUB Sample
PRINT "Start",0,0
```

```
GOSUB middle
PRINT "End",0 ,30
END

// A SUB Function called 'middle'
SUB middle: // Define the Sub
    PRINT "Middle",0,20
    RETURN // Jumps back to next command after the GOSUB call
ENDSUB // End Sub Marker
```

Output: Start Middle End

A good example of a SUB would be a function for loading the input data for a program.
Splitting up your code into blocks makes them easier to debug and to share common code with other programs.

Note : SUBS are always placed at the end of the main program. Between the commands ENDSUB and SUB no code is allowed, only comments are allowed. You are not allowed to use GOTO to jump your program to a jump mark not inside the SUB you are in, nor are you allowed to call a GOTO from the main program to a jump mark inside a SUB. These simple rules, once gotten used to, force a clean programming style.

FOR-Loops

Assuming you want to fill an array with '1's, how would you code this when the array is 10 fields long? 10 LET commands? This would work, but it's not very elegant, and not practical if your array contains hundreds or thousands of fields. We need a set of commands for repeatedly calling a section of code to set an array value, and to increment a counter (pointing at the new array location) at the same time.

We use the FOR and NEXT commands to define the code section to be repeated, and a variable to store the 'loop' counter. Lets see an example with this For-Next Sample:

```
FOR x=0 TO 9
    LET array[x]=1
NEXT
```

In this example, the code between the FOR and NEXT commands is executed while x is counting up from 0 to 9. After each call of the NEXT command, x will automatically be incremented by 1.

Line-by-line walk-through:

```
FOR x=0 TO 9
```

Assign a variable called x with the value of 0. The following code is to be repeated until x is bigger than 9.

```
LET array[x]=1
```

Assign the value 1 to the array 'array' at position 'x'

```
NEXT
```

x is increased by 1. The code between FOR and NEXT is repeated.

Now we fill an 100x100 array with the value 1.

```
FOR x=0 TO 99
    FOR y=0 TO 99
        LET array[x][y]=1
    NEXT
NEXT
```

Loops can also run backwards - e.g. `FOR x=9 to 0 STEP -1` to start the counter at 9 and go back to 0 decrementing by 1 every time. You can increment the loop counter by values greater than 1 (e.g. `FOR x=0 to 9 STEP 2` - to make it count up by 2's) as well.

The command `STEP` define the amount that will be added to / subtracted from the counter with every call to `NEXT`.

```
FOR x=24 TO 0 STEP -5
PRINT x, 0, (x*20)
NEXT
```

Output: 24 19 14 9 4

The commands `BREAK` and `CONTINUE` allow you to leave or jump to the top of a loop before the code within the loop has completed its execution.

WHILE Loops

A `WHILE` loop will be evaluated (and repeated) as long as its defining argument is **TRUE**. e.g. We want to obtain a random number in the range 3 to 10. To do this we can pick a random number from 0 to 10 and continue picking random numbers while the random number returned is less than 3.

```
LET z=0 // z starts < 3, otherwise WHILE loop
// won't ever be evaluated!
WHILE z<3 // As long as z is < 3,
z= RND(10) // z is a random number from 0-10
WEND // Repeat.
```

Another example : You want a number from 0 to 10, but not the 5. Keep picking random numbers while the last random number picked was 5.

```
LET z=5 // z=5, otherwise WHILE loop fails
WHILE z=5 // As long as z=5
z=RND(10) // z is a random number from 0-10
WEND // Repeat.
By the way, random numbers from 3 to 10 are better calculated this way:
z=RND(7)+3 // (0 to 7) +3 = (3 to 10)
```

REPEAT Loops

A `REPEAT` loop will be evaluated (an repeated) as long as its argument defined on the `UNTIL` line is **TRUE**. The following example repeats a loop until the counter reaches 10

```
LET z=0
REPEAT
DEBUG z+"\n"
INC z
UNTIL z>=10
```

Tutorial #2 (Graphics)

General

To create the final screen image which represents each frame of your application GLBasic uses three separate image buffers (or 'surfaces'). One is used to display the final image to the monitor, one is used to draw your GLBasic graphical objects (Sprites, Polygons etc.) and the last one is used to store the 'background' for your final image.

Let's take a look at these:

1. Primary Surface - This is what you see on the screen.
2. Back Buffer - This is where all graphical commands in GLBasic write to. It's invisible, that is it's not transferred to the monitor until you call the command SHOWSCREEN.

This command takes current the back buffer and transfers it to the primary surface (the monitor).
Why not have GLBasic draw directly to the monitor (Primary Surface) rather than to a Back Buffer? If you did this you would see GLBasic processing its graphical commands individually 'on-screen' rather than all-at-once, and this would result in screen flicker.

Remember: If you don't see anything you might have forgotten to use the SHOWSCREEN command.

3. Off-screen Surface This screen can be accessed with several commands - LOADBMP, BLACKSCREEN, CLEARSCREEN, CREATESCREEN and USEASBMP. Please see the Reference Section for details on these commands

When SHOWSCREEN is called, the Back Buffer and Primary Surface are swapped. The now unusable Back Buffer will be replaced (cleaned) with the Off-screen Surface which is black when no bitmap is loaded.

GLBasic uses hidden surfaces (Double Buffering) for graphical display. If you code something in GLBasic you will see nothing before your program calls the SHOWSCREEN command. If you don't see any result from your program, the first thing to check is whether you have forgotten to add the SHOWSCREEN command to it. Think of Double Buffering as two sheets of paper, one on top of the other where you draw on the bottom sheet. Once you're finished drawing, you swap the sheets and in the process erase the new bottom sheet. This hides the actual drawing of the sheet from the user and in the process eliminates screen flicker.

A game mostly is structured like this:

```
// My game
start:
LOADBMP "Background.bmp"
// Load further graphics as sprites...
// load level, setup (lives=3, time=100...)
maingame:
//Keyboard input
//Move and draw player
```

```
//Move and draw enemies
SHOWSCREEN
GOTO maingame
```

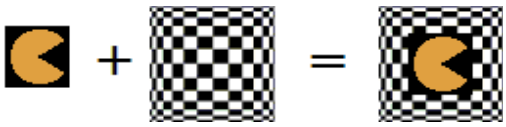
Sprites

What is a sprite? A sprite is usually a small graphic object that moves around the screen. As an example, in the game Pacman, Pacman and the Pac Ghosts are sprites and the maze is the background.

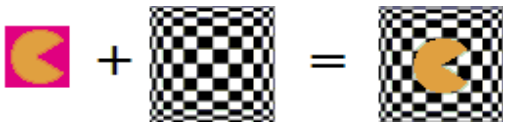
Here is a bitmap containing what could be sprites for a Pacman-style game :

If you know anything about computers you might know the truth - the PC doesn't have real sprites. Real 'sprites' in the days of the Commodore 64, Amiga, Atari XL or Archimedes, were little separate pieces of video memory that you could place an image in, these images were 'hardware overlayed' onto the display memory without damaging the background image. The PC does not have sprite hardware overlay technology but the good news is that GLBasic can emulate sprite hardware.
Sprites have to be what is referred to as "cookie cut" - i.e. the area around the shape of the sprite must be transparent. Without cookie cutting the sprite there would be a solid black rectangle around it when it was displayed.

Like this :



How does the program know which parts of the sprite are to be displayed transparently? There are many ways depending on the program you are using. GLBasic uses the following: Every pixel of an image loaded for a sprite that has the colour value red:255, green:0 and blue:128 (Hex: FF 00 80) (a shade of Pink) is considered to be transparent.



Note : All graphic files loaded are in the Windows-Bitmap format with 8 or 24 bit colour depths, uncompressed. You cannot load 16-colour graphics. You have to reload and save them as either 8-bit or

24-bit colour images.

You can also use PNG files. They use either `RGB(255,0,128)` for transparency or the alpha channel if it exists.

Your first sprite:

```
////////////////
// SPRITES
////////////////
// load sprite image from file "Bubble.bmp" and assign it to ID 0
LOADSPRITE "Bubble.bmp" ,0
// show sprite image with ID 0 at position x=300, y=150
DRAWSPRITE 0, 300, 150
SHOWSCREEN
END
```

As you can see with just 3 commands you produced a visible result. Short, sweet and simple, all the technical nitty gritty is hidden from the user by the simplicity of GLBasic. If you were to try to do this in C++ for example, the code would several pages long.

The `LOADSPRITE` command is fairly obvious. It loads a BMP file, which can be used as a sprite and assigns an ID (in this case 0) to it. Every sprite, needs a unique ID number, though the choice of ID is up to you. In the rest of the program this sprite will be referred to by its ID ('0' in this case). See also: the `GENSPRITE()` command.

Take a look at the .BMP file. You'll see there's a pink colour around the image. This part of the image will be the its transparent pixels. To demonstrate this, we'd better load an image for the background by adding...

```
LOADBMP "back.bmp"
```

...to the beginning of the program.

The command `DRAWSPRITE` needs 3 parameters. The first is the ID of the sprite to be shown. Parameters 2 and 3 are the screen position of the sprite (X,Y coordinates).

`ALPHAMODE` lets you specify an alpha value. GLBasic supports Alpha Blending. Never heard of 'Alpha Blending'? No problem...

Alpha Blending is when 2 images (namely the background and the sprite) are 'blended' together. GLBasic has two 'blending modes' for doing this.

Additive Alpha Blending

A value from 0.001 to 1.0 mixes the 2 colour values of each pixel by adding the red, green and blue values (a value over 255 (or FF in Hexadecimal) is considered to be 255). It sounds complicated, but all work is done by GLBasic. A value more than +1 for the Alpha Blending is for effects like fire, explosions and glass screens. If you want two images to get brighter when they are displayed over each

other then use this 'blending mode'.

Interpolated Alpha Blending

A value from -0.001 to -1.0 melts the 2 objects by adding both values and dividing them by 2. You can use this for shadows and other nice effects.

A tip: Have a play with the different Alpha Blending modes, try some different shapes and colours, add a few different backgrounds. Even black and white background can create a great look with coloured sprites and Alpha Blending. Play!

Some Special Sprite Commands

Zooming a sprite to any size can be done with:

```
ZOOMSPRITE id, xstart, ystart, x%, y%
```

or:

```
STRETCHSPRITE id, xstart, ystart, width, height
```

Rotating a sprite around its centre can be done with the following:

```
ROTOSPRITE id, x, y, angle
```

'X' and 'Y' values are the positions of the sprite when not rotated. So the following code:

```
ROTOSPRITE 1,50,70, 0
```

is identically to:

```
DRAWSPRITE 1,50,70
```

Both combined:

```
ROTOZOOMSPRITE id, x, y, angle, size
```

'Size' is a value that specifies the scaling of the sprite. '0.5' is half the size. '2.0' is double scaling factor.

Special General Graphic Functions

GLBasic has a command to fade from the current back buffer into a .BMP file. Fading to a black bitmap causes the screen to fade-out. This is an excellent effect for screen transitions.

```
BLENDSCREEN bmpfile$
```

GLBasic can play videos (Windows only) in all known installed video codecs. (AVI, MP2...) Simply use :

```
PLAYMOVIE filename$
```

Vector Graphic Commands

Sometimes you may want to draw something using basic shapes like lines and rectangles rather than using a sprite.

GLBasic from version 5.10522 onwards features commands for the drawing of filled rectangles, lines and single pixels. See the command reference for:

SETPIXEL - DRAWLINE – DRAWRECT

See also: LOADBMP - SETPIXEL - DRAWSPRITE

Tutorial #3 (Input)

GLBasic offers 2 types of keyboard interaction.

Text Input and Key Press Checking.

Text Input...

```
INPUT name$, x, y;
INPUT number, x, y;
```

The INPUT reads in a line of text from the keyboard until the Enter key is pressed, and then stores it in a string variable or a numeric variable. The variables 'x' and 'y' sets the initial position of the input cursor. Your program will stop until the text input is complete.

Key Press Checking...

```
IF KEY(keycode%) = 1; PRINT "KEY PRESSED", 20, 20;
```

Key press checking is used to see if a key is held down at that particular point. You need to query the output of 'Key (key code)' to see whether the key corresponding to "keycode%" is depressed.

To find out which key code corresponds to which key you can use the "Keycodes" program which resides in the "Tools" menu of the GLBasic IDE.

For example :

```
IF KEY(57) = 1; PRINT "SPACE BAR KEY PRESSED", 20, 20;
```

Mouse:

```
MOUSESTATE mx, my, ma, mb;
```

MOUSESTATE returns all the current mouse parameters.

mx = The mouse cursors screen x coordinate
my = The mouse cursors screen y coordinate
ma = The button state of mouse button A (0 or 1)
mb = The button state of mouse button B (0 or 1)

e.g.

```
MOUSESTATE mx, my, ma, mb;
if ma = 1; PRINT "Mouse button 1 is down", 20, 20;
```

`v=MOUSEAXIS(MouseFeature)`

This command returns information about a specific feature from the mouse. This information is stored, in this case, within the variable 'v'. To select which mouse feature you are interested in you pass an integer parameter to the `MOUSEAXIS` command (0 to 5 in "MouseFeature" in the above example). So if I wanted to know if the mouse wheel was moving up or down I would use the following command :

```
WHILE TRUE
v=MOUSEAXIS(2)
SELECT v
 CASE 0
  PRINT "The mouse wheel is NOT moving", 20, 20
 CASE 1
  PRINT "The mouse wheel is moving UP", 20, 20
 CASE -1
  PRINT "The mouse wheel is moving DOWN", 20, 20
ENDSELECT
WEND
```

For the full list of the parameters that MOUSEAXIS accepts, please refer to the command reference.

Joystick:

`JOYSTATE jx, jy, buttona, buttonb;`

Gets the current joystick state and writes its state to the variables. e.g. :

```
// End with ESC
WHILE TRUE // Endless loop
   JOYSTATE jx, jy, ba, bb
    a$="JX:" + jx + " JY:" + jy + " ButtonA:" + ba + " ButtonB:" +jb
    PRINT a$, 10, 100
    SHOWSCREEN
WEND
```

Tutorial #4 (Sound & Music)

This is a quite short chapter as sound playback is a relatively simple affair. You handle sounds in a similar fashion to sprites, each sound is associated to an ID.

There are 3 main commands for sound playback.

```
LOADSOUND "Boom.wav", 0, 1
```

LOADSOUND : This command loads a sound ('Boom.wav')to memory, assigning an ID ('0'). The ID is just like the ID used with the LOADSPRITE command. From that point forward you use the ID to access the sound.

The last parameter is the Buffers parameter, this defines how many channels will be opened up for this sound (polyphony). Be aware that the size of each buffer is equal to the length of the sound.

The only supported sound format is Wave-PCM.

```
PLAYSOUND (n, pan, vol)
```

This plays a loaded sound. The command requires 3 arguments.
1st : The ID of the preloaded sound
2nd : The stereo-fading. -1.0 means left, 0 = centre and +1.0 is right side.
3rd : The volume. 0.0=off, 1.0=loudest.

```
HUSH
```

This command immediately stops all sound playback.

Music

You can use music playback in one of the following formats : mp3, wav, midi.
To interact with music use the following commands:

Playback:

```
PLAYMUSIC file$
```

`PLAYMUSIC "mytune.mp3"`, will start the playback of the MP3 file "mytune.mp3".

Stopping music is done by :

```
STOPMUSIC
```

Tutorial #5 (Networking Games)

The NET... commands give network functionality to GLBasic programs using the TCP protocol. TCP has the advantage over other protocols that sending and receiving of network messages is ensured.

```
NETHOSTGAME(tcp_port%)
NETJOINGAME(server$, tcp_port%)
```

With the NETHOSTGAME command, you can host a new network server.

The value for tcp_port% specifies the TCP network port you wish to use for communication. If you set it to 0, it will use the default tcp port of 27910.

Notes :

- Ports under 1024 are known as "privileged" ports and generally support important network functions (like email and internet traffic). As a result it is a good idea to ensure you choose a port for your project higher than 1024 (maximum is 65535).
- Of course, you would use the ports under 1025 if you were specifically writing an application to connect to a service like email or a web server.
- Linux applications may need root access to open ports 1-1024.
- If you are having trouble getting client systems to talk to your server, ensure that the port chosen here is allowed through any firewalls that may exist.

```
NETSHUTDOWN
```

This command ends all NET... commands, closes any open connections and releases any memory used.

```
ok% = NETISACTIVE()
```

Returns whether the network engine is currently running. If the server quits, the client engines will be shut-down too.

```
id% = NETCREATEPLAYER(name$)
```

Creates a player that gets announced to the network server. The network services use "players" for communicating with the server and with other clients.
In the case of an error, 0 is returned.

```
num% = NETNUMPLAYERS()
```

This command returns the number of players the server is currently aware of.
If the server is stopped, this command returns 0. Note : this command performs a query on the network server, thus it's rather slow.

```
quitid% = NETGETQUITPLAYER()
```

Returns the ID of a player deleted by NETDESTROYPLAYER() or lost due to a disconnection of a client.

This command only works within the server program. If no player has quit, the return value is 0.
If a player has quit, the server should send a message to all clients informing them about the loss. You should continue to call this function until it returns 0 otherwise you may miss a disconnection if several players are removed simultaneously.

```
id% = NETGETPLAYERID(index%)
```

You can use this command for all indices in the range 0 to NETNUMPLAYERS()-1 to get the player IDs.
In case of an error, 0 is returned.

```
name$ = NETPLAYERNAME$(id%)
```

Once you have the ID for a player, you can query their display name with this function.

```
ip$ = NETGETIP$()
```

This command gets you a list of the IP addresses that are allocated to your computer, so you can tell others what they are.
The IPs are separated by "|" and can be separated with SPLITSTR.
Note : If you are writing an application to talk to systems across the internet rather than over a local network, the IP addresses returned from this command may not be the right ones for other systems to connect to. You may need to check your internet router to see what your internet facing IP address is.

```
NETALLOWJOINING OK%
```

In order to prevent more people from connecting to the server or to prevent existing computers from creating additional players, you can set NETALLOWJOINING FALSE. You must call this function in the server program.
Calling NETHOSTGAME sets this back to **TRUE**.

```
NETDESTROYPLAYER id%
```

This command removes a player from the server. This command must be called from the machine that created the player.

```
ok% = NETSENDMSG(id_from%, id_to%, text$)
```

Sends a message from player id_from% to the message mailbox of player id_to%. If id_to% = 0, the message gets sent to all known players (except the sender).

text$ is the text string to send and should not exceed 1023 characters.

```
msg$ = NETGETMSG$(id%)
```

With this command you retrieve a message for player id% from their message mailbox that was sent with NETSENDMSG. The player id% must be one you've created - i.e. you can only retrieve your own messages. If no message is available, LEN(msg$)=0.
The message is deleted after it has been retrieved.

```
id% = NETGETSENDER()
```

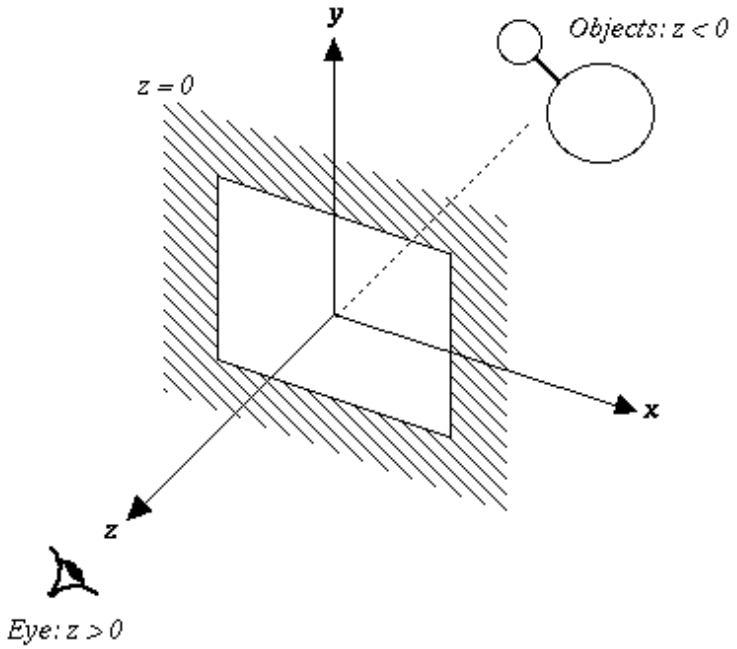
Returns the player ID of the sender of the last message received from NETGETMSG$().

```
err$ = NETGETLASTERROR$()
```

Returns an error number followed by a readable error message for the last failed action. You can convert the error into a number by calling ierr% = INTEGER(err$). It not ensured that error numbers are identical on all operating systems.

Tutorial #6 (3D Graphics)

3D graphics is a complex issue. In GLBasic we tried to make the commands simple and easy to use like the 2D commands.



If you have not completed at least one 2D project, it is advisable that you do, 3D is a large step up from 2D. Jumping from compiling your first program directly into a 3D project could be an impossibly steep step for many new users.

GLBasic uses what is known as the right hand system - where Y always points upwards.

All 3D commands start with X_... so that you might find them very easily in your program and within this reference.

The 3D commands, whilst available in the free version of GLBasic, will activate the 5 minute demo timer – for full and unrestricted use, a license should be brought.

Textures must be 2^n size (width or height must be 2, 4, 8, 16, 32, 65, 128, 256, 512...) Be aware that some models you may get from the internet will not feature this restriction. If this is the case with your models, then you will have to stretch the texture manually to an appropriate size - 64, 128, 256, 512 or 1024 pixels squared - otherwise ugly texture errors will occur.

Basically GLBasic 3D works like this:

Switching to 3D mode:

```
X_MAKE3D znear, zfar, field_of_view
```

Where znear and zfar describe the clipping range in a relative distance to the camera. Values of 1 and 1000 usually give good results. Bigger distances might cause a lack of quality when z-buffering the 3D scene. The field_of_view angle lets you specify the camera's zoom objective.
Camera

```
X_CAMERA x, y, z, px, py, pz
```

Sets the camera at position x, y, z and points it towards the point px, py, pz. Top is always along the positive Y axis.

Light :

```
X_AMBIENT_LT num, col
```

Turns on an ambient light. 'Ambient' means light that shines from every direction. 'col' specifies the colour of the light (See RGB).
'num' is the reference ID number of the light source. Values from 0-7 are allowed, since modern hardware doesn't support more than 8 lights at a time. If there are more than 9 light sources in a scene, the graphics card will try and use the 8 nearest lights. In professional applications no more than 2 lights are active within a scene at one time as it can cause the application to slow down.

```
X_SPOT_LT num, col, x, y, z, dirx, diry, dirz, cutoff
```

Creates a spot light at position x, y, z that shines in direction dirx, diry, dirz. The cutoff specifies the cut-off (width of the cone) angle of the spot light. The regular light in your scene will not (even if it seems a bit illogical in the beginning) cast any shadows. Instead the rendered triangle for the object will be drawn lighter or darker, depending on the angles between the camera, the triangle's normal, and the light's position.

Transformations

Now that the camera and lights are set, we need the most important thing, some 3D objects.

```
X_MOVEMENT x, y, z
X_SCALING fx, fy, fz
X_ROTATION phi, dx, dy, dz
```

X_MOVEMENT moves the current 3D drawing position to point x, y, z. The local origin of an object (0,0,0) will be moved to this position when drawing it.

X_SCALING scales an object factorizing the local axis. X_SCALING 0.5, 0.5, 0.5 for example halves the size of an object. X_SCALING 2, 2, 2 doubles the size of an object.

X_ROTATION rotates an object around the vector dx, dy, dz by angle phi. There can be several rotations queued up.

X_MOVEMENT or X_SCALING will reset all previously added rotations and must therefore be called before X_ROTATION.

Objects

The simplest way to create a 3D object within GLBasic is via loading a 3D file. You simply create it with a 3D program, export it in Quake '.md2' or 3D Studio '.3ds' format, and convert it to a compressed '.ddd' file with the 3DConvert tool. Once you have done this, you can load it as follows:

```
X_LOADOBJ filename$, idnum%
```

Furthermore you can build objects at runtime with these commands.

```
X_OBJSTART idnum
X_OBJADDVERTEX x1, y1, z1, tx1, ty1, col
X_OBJADDVERTEX x2, y2, z2, tx2, ty2, col
X_OBJADDVERTEX x3, y3, z3, tx3, ty3, col
X_ENDOBJ
```

idnum specifies the reference ID number of the object. With this number you can draw it later. Existing object numbers can be overwritten.

X_OBJADDVERTEX adds a vertex to an object.

tx, ty specify the texture coordinates (uv,uy - from the rang 0 to 1, 0.5 for instance is the center of the texture), '0'=top-left, '1'=bottom-right. The first 3 vectors form a triangle. Every additional vector forms a triangle with the previously last specified 2 vectors. If you don't want this method of drawing, you can 'lift the design pen' and restart with 3 independent vertices that for a new triangle by using the command.

```
X_OBJNEWGROUP
```

Now the loaded or created object must be drawn. Do this with:

```
X_DRAWOBJ idnum, frame
```

Whereby 'frame' is the index of the animation 'frame'. User defined objects have no animation, so you have to set it to '0' (i.e. the first frame)

You can play animations with :

```
X_DRAWANIM num, frame, from, upto, interpol, full_anim
```

'from' and 'upto' define the start and end frames for the animation. 'interpol' specifies how smooth the animation will be (interpolation step from 0 to 1). 'full_anim' indicates, that all frames in between 'from' and 'upto' will be used for animation. If you set it to **FALSE**, the frames will be interpolated directly. See the reference page for X_DRAWANIM for further details on the 'interpol' parameter.

In order to assign a texture to your object, you must have previously loaded a texture with LOADSPRITE or LOADBUMPTEXTURE and assign the texture with the X_SETTEXTURE command.

```
X_SETTEXTURE num, num_bump
```

'Num' represents the Sprite ID number, -1 indicates no texture should be used.
The parameter num_bump# specifies which normal map should be used for Dot3 Bump Mapping. The normal map can be created using LOADBUMPTEXTURE instead of LOADSPRITE. A value of -1 specifies that no bump mapping should be performed.

Finishing 3D

When working in 3D mode you cannot use any of the 2D commands. In order to use 2D commands you will have to change back to 2D mode with X_MAKE2D.

```
X_MAKE2D
```

Special Rendering Modes

If you want to use "Cel Shading", "Bump Mapping" or "Stencil Shadows", see the X_SPOT_LT function.

With X_SPHEREMAPPING you can program metallic reflections.

Tutorial #7A - Types Made Simple

A TYPE in GLBasic is just a way of grouping related variables together - essentially a composite data type containing variables of other variables (including TYPE)

With V8, a TYPE can also contain functions, which can access variables contained in the TYPE Contents

TYPE Declaration

A TYPE is defined in the following way :

```
TYPE name
    [variable1]
    ...
    [variableN]

     or

    [variable1][;variableN]

    [function1]
    ...
    [functionN]
ENDTYPE
```

name is valid name, that you need to give to define the TYPE
variable1 to **variableN** define the variables that you want to be contained in the TYPE (if any), whilst **function1** to **functionN** define the functions in the same TYPE

Note :
You need at least 1 variable OR at least 1 function, otherwise GLBasic will through up a compiler error

Type Creation

There are several ways of creating a type. If you know exactly how many indices there will be and the amount will rarely change, you could use the DIM statement :

```
LOCAL example[] as tExample; DIM example[4]
```

However, the preferred way is to use DIMPUSH. Whilst this seems a bit more complicated to use, it could be regarded as giving you a bit more freedom of ordering. An example would be :

```
LOCAL example[] as tExample
LOCAL record as tExample
record.info%=1
DIMPUSH example[],record
```

This puts adds a new record to the end of the **example[]** array.

Accessing members in a TYPE

To access a member in a TYPE, you use a dot (.) operator, for example :

```
DEBUG example.info%
```

This method of access is just the same if you use arrays of types :

```
DEBUG example[4].info%
```

Sorting

A TYPE can be sorted (see the SORTARRAY and ADDRESSOF commands for more information), but a short example is :

```
SORTARRAY example[],0
```

This will sort the array based on the first variable in the type, and will sort in ascending order.
For a custom sort, you would use something like :

```
SORTARRAY example[],ADDRESSOF(sortRoutine)
```

And sortRoutine would look like :

```
FUNCTION sortRoutine:a as tExample,b as tExample
    IF a.value% < b.value% then return 1
    IF a.value% > b.value% then return -1
    RETURN 0
ENDFUNCTION
```

This function would sort the array in ascending order

Deleting a TYPE

There are several ways of deleting a TYPE in an array. You can either use `DIMDEL` if you know the correct indices to delete or use `DELETE` if you are using the `FOREACH` command. For example :

```
DIMDEL example[],0 - To delete the first type in an array.

FOREACH loop IN example[]
    DELETE loop
NEXT
```

Which would delete each record in the **example[]** array. Note that with `DELETE` in a loop, no code will be executed after this as the loop will move to the next item in the array straight away.

You cannot delete a single type, so there is no way of deleting something like :

```
LOCAL a as tExample
```

However, in a FUNCTION or SUB it will only last until the routine has finished.

Passing Types

Passing types to functions can be done quite easily. Types are passed *by reference*, so any modifications made within the function will be available to the routine calling the function. Passing a type is define line :

```
FUNCTION test:a as tExample
```

with **a** defining a variable of type tExample. The variable **a** doesn't have to correspond to the original variable using the type.

A more detailed example of this is :

```
TYPE tTest
        a%
ENDTYPE

LOCAL a AS tTest

a.a%=12
DEBUG "Value of a.a%="+a.a%+"\n"
change(17,a)
DEBUG "Value of a.a%="+a.a%+"\n"

FUNCTION change%:amount%,b AS tTest
        INC b.a%,amount%
ENDFUNCTION
```

This produces an output of :

Value of a.a%=12
Value of a.a%=29

Passing variables into functions saves having to define variables as a GLOBAL, and thus cutting down the changes of programming errors occurring by modifying a global at the wrong time (or place).

Returning Types

Types can be returned from a function by using the TYPE name after the function name, like :

```
FUNCTION change AS tTest:amount%
```

Types returned back are done by value – that means a copy a copy of the TYPE is returned, rather than the original.

An example of returning Types :

```
TYPE tTest
        a%
ENDTYPE

LOCAL a AS tTest
LOCAL b AS tTest

a.a%=12
DEBUG "Value of a.a%="+a.a%+"\n"
b=change(5,a)
DEBUG "Value of a.a%="+a.a%+"\n"
DEBUG "Value of b.a%="+b.a%+"\n"

FUNCTION change AS tTest:amount%,b AS tTest
LOCAL temp AS tTest

        temp.a%=b.a%+amount%
        RETURN temp
ENDFUNCTION
```

Output :

Value of a.a%=12
Value of a.a%=12
Value of b.a%=17

Tutorial #7B - Using Extended Types

As mentioned previously, types can be used to group variables into an abstract object to make it easier to manipulate information for a specific purpose - instead of having the variables defined all around the place...

It is therefore a logical step to be able to group functions into a type - this allows functions to be created to the specific type and to allow easy manipulation of its variables. It keeps things tidy, aid easier code re-use and help debugging too. Extended types are just like a TYPE, but with functions in them.

Extended types could be described as being like a C++ class, but with some differences :

Each type has to have unique function names

No private or protected variables or functions - this mean any function (both within and outside a type) can call a function (or variables) within a type

self

This command is needed to access the variables and functions within a type - without this you could get some strange and complicated compiler errors.

Name Defining

Functions have to be uniquely named in each type, but each type can have the same function names, although this wasn't allowed in the V8 beta

It would be a good idea to define a type depending on its contents, to allow easier debugging, testing and code reading.

If a type wont have any functions in it, append a lower-case 't' to the name. If the type will contain functions, user an upper-case 'T' before the name. See *The Types* section below for examples.

Function and Variable Access

As functions and variables can be directly accessed, it would be best not to do so, and instead have helper functions. There are many advantages of doing this : easier to debug, put all validation code into one place and the knowledge that you wont be causing extra problems.

Constructor and Destructor

As there isn't the ability to call a function when the type is created or destroyed, the former can be done by calling a function (I usually call this **Initialise**) just after the function is defined.

For a destructor, it might be a good idea to create a function called Destroy, which can then be called if any locally allocated sprite, 3D graphics or sound ID's are used.

The Types

A standard type could be :

```
TYPE tSomething
    a%
    b
    c$
ENDTYPE
```

whilst an extended type can be defined as :

```
TYPE TDot
  value
  FUNCTION TDot_Initialise%:
        self.value=123.45
  ENDFUNCTION
ENDTYPE

TYPE TSomething
  a%
  b
  c$
  dot AS TDot

  FUNCTION TSomething_Initialise AS TDot:
        dot.TDot_Initialise()
        RETURN self.dot
  ENDFUNCTION
ENDTYPE

LOCAL some AS TSomething
LOCAL _dot AS TDot

_dot=some.TSomething_Initialise()
DEBUG _dot.value
```

This example returns the TDot extended type as created in the TSomething class

When defining parameters for functions, be aware the the variables aren't in the local space (yet), so something like :

```
TYPE TDot
  value
  FUNCTION TDot_Initialise%:x%,y%
        self.value=x%*y%*1.2
  ENDFUNCTION
ENDTYPE

TYPE TSomething
  a%
  b
  c$
  dot AS TDot

  FUNCTION TSomething_Initialise AS TDot:x%,y%
        dot.TDot_Initialise()
        RETURN self.dot
  ENDFUNCTION
ENDTYPE
```

```
GLOBAL x%,y%
LOCAL some AS TSomething
LOCAL _dot AS TDot

	_dot=some.TSomething_Initialise(1,2)
	DEBUG _dot.value
```

will produce the following warnings :

"Test.gbas"(20) warning : variable already exists : x
"Test.gbas"(20) warning : variable already exists : y

This shows that you can modify a passed type :

```
TYPE TDot
  value
  FUNCTION TDot_Initialise%:x%,y%
	self.value=x%*y%*1.2
  ENDFUNCTION

  FUNCTION TDot_Display%:
	DEBUG self.value+"\n"
  ENDFUNCTION
ENDTYPE

TYPE TSomething
  a%
  b
  c$
  dot AS TDot

  FUNCTION TSomething_Initialise:d AS TDot
	self.dot=d

	self.dot.TDot_Initialise%(1,2)
	self.dot.TDot_Display()

	d.TDot_Initialise(6,7)
  ENDFUNCTION
ENDTYPE

GLOBAL x%,y%
LOCAL some AS TSomething
LOCAL _dot AS TDot

_dot.TDot_Initialise(2,3)
_dot.TDot_Display()

some.TSomething_Initialise(_dot)
_dot.TDot_Display()
```

And, of course, they can be in arrays. This is handy for, say, multi-player games, as you wont need to pass things like player indexes to functions :

```
TYPE TDot
  value
  FUNCTION TDot_Initialise%:x%,y%
	self.value=x%*y%*1.2
  ENDFUNCTION

  FUNCTION TDot_Display%:
	DEBUG self.value+"\n"
  ENDFUNCTION
ENDTYPE

TYPE TSomething
  a%
```

```
  b
  c$
  dot AS TDot

  FUNCTION TSomething_Initialise:d AS TDot
        self.dot=d

        self.dot.TDot_Initialise%(1,2)
        self.dot.TDot_Display()

        d.TDot_Initialise(6,7)
        d.TDot_Display()
  ENDFUNCTION
ENDTYPE

GLOBAL x%,y%
LOCAL some[] AS TSomething
LOCAL _dot AS TDot

DIM some[65]
some[0].TSomething_Initialise(_dot)
```

ADDRESSOF

ADDRESSOF can't be used with a function in an extended type. This shouldn't matter as all the function would be doing is comparing the contents of two normal types.

Multiple Files

With V8.050 upwards, types can be created in one file and used in another.

Returning Types

From V8.148, extended types can now return types, making the following possible :

```
TYPE TMoo
        a%

        FUNCTION go%:
                DEBUG "Output"
        ENDFUNCTION
ENDTYPE

TYPE TTest
        FUNCTION yes AS TMoo:
        LOCAL moo AS TMoo

                moo.a%=12
                RETURN moo
        ENDFUNCTION
ENDTYPE

LOCAL t AS TTest
LOCAL p AS TMoo

        p=t.yes()
        DEBUG "Value of a% : "+p.a%+"\n"
```

You can also complicate things by storing a result in a variable that originally called a function :

```
TYPE TTest
        a%

        FUNCTION yes AS TTest:
        LOCAL moo AS TTest
```

```
                self.a%=16
                moo.a%=19
                RETURN moo
        ENDFUNCTION

        FUNCTION exit%:
                RETURN self.a%
        ENDFUNCTION
ENDTYPE

LOCAL t AS TTest

        t=t.yes()
        DEBUG "t.a% ="+t.a%+"\n"
        DEBUG "t.a% = "+t.exit()+"\n"
```

Both debug statements return 19, as the local 't' variable is replaced by that returned by the 'yes' function.

This sort of this is safe, as GLBasic will not leak memory. What happens here is :

```
LOCAL t AS TTest – one instance of Ttest

t.yes()   // internally creates another instance (moo) of TTest
t=t.yes() // The local moo is copied over the t.
          // After the copy the local moo will be dropped from the
          // stack.
```

Tutorial #8 (Your First Game)

One More

About the Game

The original game (I think) was part of a game called LOGO by Starbyte for the Amiga500*. The aim is to remove all the coloured dots from a play-field. Whenever the player clicks a block, that block and the blocks to either side, above, and below toggle their state. Thus, clicking the centre point of this play-field:

```
###
###
###
```

Would cause it to look like:

```
#+#
+++
#+#
```

This game is pretty easy to program and leaves a lot of space for your own ideas to finish it. It's also nice since we can have computer generated levels. We simply have the computer randomly click some points. It's rather difficult to play once the computer has clicked more than 10 times or so...

New Project

After starting GLBasic, select the button "new project" and give it a name and a path for the project. e.g. "..\GLBasic\Projects\OneMore"

The Main Program

The basic code will look like this (pseudo-code):

```
main:
IF level_complete THEN create_new_level
IF mouse_click THEN change_clicked_stones
show_playfield
GOTO main
```

First we need a playfield to store whether a block is activated or deactivated (cleared). We use the value 0 for deactivated since the DIM command always sets all values to 0 automatically. Thus, the first IF statement will be evaluated and a new level will be created for us.

```
// ------------------------------- //
// Project: OneMore

GLOBAL playfield%[], level
LOCAL mx, my, b1, b2
DIM playfield[10][10]
level = 0
```

Now, we have an array called "playfield" that is 10x10 numbers in size. We can address them with the indices 0-9. See the DIM command.

```
// Main game
main:
MOUSESTATE mx, my, b1, b2
PRINT "<=", mx, my-8
SHOWSCREEN
GOTO main
END
```

Make sure that you have Debug Mode enabled:

or

Compile and run it. It looks nice, but it doesn't do much. We have a mouse pointer that we can move using the mouse. For simplicity, the program is extremely minimal. To end/close/quite the program, press the ESC key.

Take a look at the reference for all the commands we used here. The `PRINT` line is our mouse-pointer. You can load and use a nice sprite image for it later. The my-8 is because the font is 16 pixels high, and I subtracted half of it. Don't forget the `SHOWSCREEN`!

Showing the Playfield

Now we want to show the playfield data so we add a new subroutine (SUB). Press the SUB-button:

and enter the name: ShowPlayfield

This creates an empty "skeletal" subroutine. Make sure it is at the end of your source file (not somehow in the middle of any pre-existing code) and complete it like this:

```
// ------------------------------------------------------------ //
// ---  SHOWPLAYFIELD  ---
// ------------------------------------------------------------ //
SUB ShowPlayField:
	LOCAL x, y, color[]

	DIM color[2]
	color[0] = RGB( 50, 50, 255) // Not set
	color[1] = RGB( 50, 255, 50) // Set

	DRAWRECT 0,0, 320, 320, RGB(255,255,255)
	FOR x=0 TO 9
		FOR y=0 TO 9
			DRAWRECT x*32+1, y*32+1, 30, 30, color[playfield[x][y]]
		NEXT
	NEXT
```

```
        PRINT "Clicks: "+clicks, 360, 120
        PRINT "Level: "+level, 360, 160
ENDSUB // SHOWPLAYFIELD
```

Here we create a local array called color and assign a blue color (50,50,255) to the first index [0] and a green color (50,255,50) to the second index [1]. Next, a white rectangle gets drawn that blanks out the playfield area.

We then do a double FOR loop, One for x and one for each y of each x. In this loop we draw a filled rectangle for each cell of our playfield-array. We use color[playfield[x][y]] for the RGB value. Thus a zero in that array location causes a blue coloured rectangle, a 1 causes a green coloured rectangle.

Also we inform the user about the current level and the number of times they have clicked during this level.

Add this line to the main game below the "MOUSESTATE" line:

```
GOSUB ShowPlayfield
```

and add "clicks" to the GLOBAL variable declarations:

```
GLOBAL  playfield%[], level, clicks
```

then run the game. It looks better now, but there's not much interactivity here, right?

Interactivity

Next, add a function with the FUNCTION button:

and give the name "Change" and the arguments: x, y

This is the point where we want to change the playfield array. Complete it as follows:

```
// ------------------------------------------------------------ //
// ---  CHANGE  ---
// ------------------------------------------------------------ //
FUNCTION Change: x, y
        // These values are defined LOCAL:
        // x,  y
        // x, y
        LOCAL set
        IF x>=0 AND x<10 AND y>=0 AND y<10
        set = playfield[x][y] // Get old state

        IF set = TRUE
                set=FALSE
        ELSE
                set=TRUE
        ENDIF
        // or simply:
        // set = 1-set
        playfield[x][y]=set // Set new state
        ENDIF
ENDFUNCTION // CHANGE
```

In this function we will try to make the playfield[x][y] be 1 if it was 0 and vice versa. Before we change it though, we check if the point (x, y) is out of the 0-9, 0-9 bounds. If it is out of this range, we don't change anything or we will get an error from the running program: DIM out of range. This error means we've tried to update a value in our array which doesn't exist (the position we tried to update is "out of (the) range" of the array.

Now add another function with the FUNCTION button. Give it the name "Click" and the argument list: x, y This is the point where the player clicks will be processed.

```
// ------------------------------------------------------------ //
// ---  CLICK  ---
// ------------------------------------------------------------ //
FUNCTION Click: x, y
	// These values are defined LOCAL:
	// x,  y
	LOCAL clicks
	clicks=clicks+1
	Change(x-1, y)
	Change(x+1, y)
	Change(x, y)
	Change(x, y-1)
	Change(x, y+1)
ENDFUNCTION // CLICK
```

Here, we Change() the value of the clicked point, as well as the points surrounding it. Also we increase a variable (clicks) indicating how often the player has clicked in this level.

Now change the main program, so it looks like:

```
GLOBAL  playfield%[], level
LOCAL mx, my, b1, b2, mousefree

DIM playfield[10][10]
level = 0

// Main game
main:
MOUSESTATE mx, my, b1, b2
IF b1
	IF mousefree THEN Click(mx/32, my/32)
		mousefree=FALSE
ELSE
	mousefree=TRUE
ENDIF

GOSUB ShowPlayField
PRINT "<=", mx, my-8
SHOWSCREEN
GOTO main
END
```

The mousefree variable tell us whether the user is still holding the mouse button down from the last time we processed the Click() function. If the player is still holding the button down, we don't reprocess the click again and again, but wait until they have released the mouse and then re-clicked it. Now run the program. It should work nicely now. Get a feeling for how the game works. Click randomly a few times and try to clear the board again. Is it fun?

Levels

Add a new SUB called "NewLevel":

```
// ------------------------------------------------------------ //
// ---  NEWLEVEL  ---
// ------------------------------------------------------------ //
SUB NewLevel:
	LOCAL x, y, i

	// Clear playfield - just to be sure
	FOR x=0 TO 9
		FOR y=0 TO 9
			playfield[x][y]=FALSE
		NEXT
	NEXT

	level=level+1
	// Do random clicks
	FOR i=0 TO level
		Click(RND(9), RND(9))
	NEXT
	clicks=0
	GOSUB ShowPlayField
	PRINT "Level: "+level + " - get ready", 0, 360
	SHOWSCREEN
	MOUSEWAIT
ENDSUB // NEWLEVEL
```

In this subroutine, we first clear the playfield to really make sure it's cleared. Next we increase the level variable. The computer then clicks some random points (more are clicked as the level being played increases) in the playfield and we reset the number of clicks occurred back to 0.

The computer can now create a level but it doesn't know when the level is cleared. We need a function that tells us when this has occurred. Add a function called "IsLevelComplete" with no parameters:

```
// ------------------------------------------------------------ //
// ---  ISLEVELCOMPLETE  ---
// ------------------------------------------------------------ //
FUNCTION IsLevelComplete:
	LOCAL x, y

	FOR x=0 TO 9
		FOR y=0 TO 9
			IF playfield[x][y]=FALSE THEN RETURN FALSE
		NEXT
	NEXT
	RETURN TRUE
ENDFUNCTION // ISLEVELCOMPLETE
```

Here we simply loop through all the locations in our playfield and return **FALSE** if any of the playfield's cells is set to 1 (=**TRUE**). If the loop finishes without returning from the function, there must be no more green dots left, thus the board is clear and we return **TRUE** at the end of the function.

In order to test the whole game insert this code:

```
IF IsLevelComplete()
	GOSUB NewLevel
	mousefree=FALSE
ENDIF
```

to the main program after the main: jumpmark. Run the game. How does it feel to have written a

complete computer game? If you add a fee images called "image0.bmp", "image1.bmp", "image2.bmp" and so on to the "OneMore\OneMore.app" folder and add the line:

```
BLENDSCREEN "images"+level+".bmp"
```

to the start of the NewLevel subroutine, the game is even better than the original Amiga game was back then! Doesn't that feel great? You are the master of the computer now - you tell it what to do and it follows your commands. You can create worlds with a few lines of code.

The Completed Program

Here's the complete source code in case you might have missed something:

```
// ------------------------------- //
// Project: OneMore

GLOBAL  playfield%[], level, clicks
LOCAL mx, my, b1, b2, mousefree

DIM playfield[10][10]
level = 0

// Main game
main:
IF IsLevelComplete()
        GOSUB NewLevel
        mousefree=FALSE
ENDIF
MOUSESTATE mx, my, b1, b2
IF b1
        IF mousefree THEN Click(mx/32, my/32)
                mousefree=FALSE
ELSE
        mousefree=TRUE
ENDIF

GOSUB ShowPlayField
PRINT "<=", mx, my-8
SHOWSCREEN
GOTO main
END



// ------------------------------------------------------------ //
// ---  SHOWPLAYFIELD  ---
// ------------------------------------------------------------ //
SUB ShowPlayField:
        LOCAL x, y, color[]

        DIM color[2]
        color[0] = RGB( 50, 50, 255) // Not set
        color[1] = RGB( 50, 255, 50) // Set

        DRAWRECT 0,0, 320, 320, RGB(255,255,255)
        FOR x=0 TO 9
                FOR y=0 TO 9
                        DRAWRECT x*32+1, y*32+1, 30, 30, color[playfield[x][y]]
                NEXT
        NEXT
        PRINT "Clicks: "+clicks, 360, 120
        PRINT "Level: "+level, 360, 160
ENDSUB // SHOWPLAYFIELD



// ------------------------------------------------------------ //
// ---  CHANGE  ---
```

```
// ------------------------------------------------------------ //
FUNCTION Change: x, y
	// These values are defined LOCAL:
	// x,  y
	// x, y
	LOCAL set
	IF x>=0 AND x<10 AND y>=0 AND y<10
	set = playfield[x][y] // Get old state

	IF set = TRUE
		set=FALSE
	ELSE
		set=TRUE
	ENDIF
	// or simply:
	// set = 1-set
	playfield[x][y]=set // Set new state
	ENDIF
ENDFUNCTION // CHANGE

// ------------------------------------------------------------ //
// ---  CLICK  ---
// ------------------------------------------------------------ //
FUNCTION Click: x, y
	// These values are defined LOCAL:
	// x,  y
	clicks=clicks+1
	Change(x-1, y)
	Change(x+1, y)
	Change(x, y)
	Change(x, y-1)
	Change(x, y+1)
ENDFUNCTION // CLICK

// ------------------------------------------------------------ //
// ---  NEWLEVEL  ---
// ------------------------------------------------------------ //
SUB NewLevel:
	BLENDSCREEN "images"+level+".bmp"
	LOCAL x, y, i

	// Clear playfield - just to be sure
	FOR x=0 TO 9
		FOR y=0 TO 9
			playfield[x][y]=FALSE
		NEXT
	NEXT

	level=level+1
	// Do random clicks
	FOR i=0 TO level
		Click(RND(9), RND(9))
	NEXT
	clicks=0
	GOSUB ShowPlayField
	PRINT "Level: "+level + " - get ready", 0, 360
	SHOWSCREEN
	MOUSEWAIT
ENDSUB // NEWLEVEL



// ------------------------------------------------------------ //
// ---  ISLEVELCOMPLETE  ---
// ------------------------------------------------------------ //
FUNCTION IsLevelComplete:
	LOCAL x, y

	FOR x=0 TO 9
		FOR y=0 TO 9
			IF playfield[x][y]=FALSE THEN RETURN FALSE
		NEXT
	NEXT
	RETURN TRUE
ENDFUNCTION // ISLEVELCOMPLETE
```

Exercise

There's a logical flaw in the program as given, in that as the board does not take up the whole screen then the user can click outside the playing field to the right or below, and toggle the state of points on the board. Can you find a way to prevent this illegal move from being played?

Also - how will you deal with running out of image files for backgrounds? What happens if you have 2 backgrounds (image0.bmp and image1.bmp) and the player gets to a third level?
according to original help file

Tutorial #9 (Homework)

Tasks for testing what you have learned

That has been quite a lot of information to learn. The coming tasks will give you a good start into your game programming life. You should take a full day for each task and fiddle around with the program after getting it to work.

General

So enough with the basics. You have learned the command `PRINT`. Here the syntax again:
`PRINT` [NUMBER/WORD], X-Pixel to the right, Y-Pixel towards the bottom;

A little test that will prepare you for the next chapter: You are a teacher and your class contains 4 pupils (2 rows, 2 columns). Since you are such a nice teacher, you want to have a computer program choose a pupil to lead the class each day. The program should give the name and the sitting position of the pupil printed to the screen. Class:

	Column 0	Column 1
Row 0	Tom	Markus
Row 1	Sabine	Manuela

Output e.g.
Sabine sitting in: Row 1 Col 0

Tips: -If you don't change the standard font, the chars are 16x16 pixels each. -A random number will be returned by the function `RND(maximum)`. (Take a look back top)

Graphics

Write a program that loads 2 sprites and does the following with them: Move sprite 1 from left to right, then rotate with `ROTOSPRITE` once, then use `ROTOZOOMSPRITE` to rotate the sprite once again and zoom it to full screen size. Repeat with the second sprite.

Now the program should load a background image and do the same routines again with an alpha value of -100 for sprite 1, and 100 for sprite 2.

Tutorial #10 (Advanced Programming)

Advanced topics:

This section is designed for experienced programmers and advanced users. You can use a lot of advanced programming techniques in GLBasic.

You can define local variables for subroutines (or the main loop) if you want to keep the subroutine separated from the main code and want to re-use it in another program. You define a local variable within a SUB by using the LOCAL keyword. You can define local numbers, strings and arrays like this:

```
LOCAL num, word$, num_array[], word_array$[]
```

To access a global variable that has the same name as a previously defined local variable use the GLOBAL keyword.

Dynamic Arrays

GLBasic offers you the ability to re-dim arrays with DIM. If you want to keep the overlapping data use REDIM.

```
DIM a[5]
...
DIM a[12][4] // Yes!!
```

Freeing Memory

Freeing memory used by resources is generally not supported in GLBasic. You can do this very simply though by loading a resource that does not exist in place of the original resource.
This will free the memory that was used to store the image of sprite '0'.

```
LOADSPRITE "xx", 0
```

This will free sound number '0'

```
LOADSOUND "xx", 0, 1
```

Most people however, clear ID's by using an empty filename :

```
LOADSPRITE “”,0
```

This will free the memory used for this array.

```
DIM a[0]
```

Recursion

Functions can be recursive. That is, a function can call itself a limited number of times. Care should be taken to make sure there is always an "exit strategy", so that the recursive loop can exit at some point. Without this (also known as a base case), your program would crash due to running out of stack space.

An example of recursion is :

```
DEBUG "Factorial : "+factorial(20)
END

FUNCTION factorial%:amount%
	IF amount%<=1
		RETURN 1
	ELSE
		RETURN amount%*factorial(amount%-1)
	ENDIF
ENDFUNCTION
```

The function calls itself recursively on a smaller version of the input (amount% - 1) and multiplies the result of the recursive call by amount%, until reaching the base case, analogously to the mathematical definition of factorial.

Hexadecimal Numbers

Hexadecimal numbers can also be used in addition to decimal numbers by prefixing it with '0x'.
ad=127; ah=0xff;

Tutorial #11 (The Second Game)

PONG

Now that the basics are clear, the skills you've learnt must be strengthened and practised. This second game is called "PONG". It is not only one of your first games in GLBasic, but it was also one of the first computer games of the world (1972 by Ralph H. Baer). If you are not familiar with PONG, you can read all about it here: http://www.pong-story.com

New Project

After the start of GLBasic select the button "new project" and give the project a name and an appropriate path for the project. e.g. ..\GLBasic\Projects\Pong

Variables

Now that the project is set up, the programming work can begin. We will need variables for the bat's x and y positions, as well as one for the score obtained so far. For all of these variables we will define a separate array so that 3, 4 or more players could easily be added later. This also allows us to reuse the code for bat movements and scoring in a loop.

More about this later...

Our code looks like this:

```
// Pong
DIM bat_y[2]
DIM bat_x[2]
DIM score[2]
```

Firstly we must initialize the players. To do this we add a subroutine (SUB) with the menu option: "project/new SUB". Give it the name "Init".

We need to write some code to draw the playing field and convert it into the Back-buffer so it will be drawn each frame. You could also load a picture rather than creating it by hand if you wanted to. Once this has been done we set the position of the racquets to the edge coordinates.

```
// ------------------------------------------------------------ //
// -=#  INIT  #=-
// ------------------------------------------------------------ //
SUB Init:
 // Draw playfield, use as background bitmap
 BLACKSCREEN
 DRAWRECT 0,0, 640, 16,  RGB(255, 255, 255)
 DRAWRECT 0, 464, 640, 480, RGB(255, 255, 255)
 DRAWRECT 312, 0, 16, 480, RGB(255, 255, 255)
 USEASBMP

 //Reset Bat Y Locations
```

```
 bat_y[0]=240; bat_y[1]=240
 // Reset Bat X Locations
 bat_x[0]=16; bat_x[1]=600
ENDSUB // INIT
```

Reset the ball

At the beginning of each game, the ball must be placed in the centre of the field and its speed must be set to 1. For this we create a new subroutine as described before and call it "ResetBall".

```
// ------------------------------------------------------------ //
// -=#  RESETBALL  #=-
// ------------------------------------------------------------ //
SUB ResetBall:
 // Diese Variablen sind als LOCAL definiert:
 // These variables are defined LOCAL:
 // void
 ball_x=320
 ball_y=240

 IF ball_sx<0
  ball_sx=1
 ELSE
  ball_sx=-1
 ENDIF

 ball_sy=1
ENDSUB // RESETBALL
```

We need to call this function in Init, thus we add

```
GOSUB ResetBall
```

to the top of the subroutine "Init".

Viewing

We have typed a lot of code already, but still there is nothing to see. It is now time to show the playing field and the ball. For this we will create a new subroutine with the name ShowAll:

```
// ------------------------------------------------------------ //
// -=#  SHOWALL  #=-
// ------------------------------------------------------------ //
SUB ShowAll:
 // Die Schläger anzeigen / Show the bats
 FOR num=0 TO 1
  DRAWRECT bat_x[num], bat_y[num], 16, 64, RGB(255, 255, 255)
  PRINT score[num], num*320 + 32, 16
 NEXT
 // Der Ball / Draw the ball
 DRAWRECT ball_x, ball_y, 16, 16, RGB(255, 255, 255)
 SHOWSCREEN
ENDSUB // SHOWALL
```

Main Loop

Now we need a main loop. The main loop will call the game's other subroutines over and over again. Add the following lines above the first subroutine in the program. Code written above the first 'SUB' or 'FUNCTION' defines the main program. This main program will be executed when the program starts.

```
// pre-settings
GOSUB Init
MainLoop:
 GOSUB ShowAll
GOTO Main
```

Beware! This code would cause the program to look like it was doing nothing if we did not call the instruction SHOWSCREEN in the ShowAll subroutine

The First Run

You can now press the compile button and start the program. (F8, then F5)

Move!

The game now has a display but it is lacking movement. To fix this we will need a subroutine called 'MoveAll'. Before adding this though, we need to make sure our main loop will call the new MoveAll subroutine.

Before the following line in the main loop :

```
GOSUB ShowAll
```

add the following line

```
GOSUB MoveAll
```

in the main loop.

Now create a new subroutine called 'MoveAll' and add the following code :

```
// ------------------------------------------------------------ //
// -=#  MOVEALL  #=-
// ------------------------------------------------------------ //
SUB MoveAll:
// Paddles
 FOR num=0 TO 1
  // Keys:  A, Z
  IF KEY(30) THEN bat_y[0]=bat_y[0]-2
  IF KEY(44) THEN bat_y[0]=bat_y[0]+2
  // Keys:  /\, \/
  IF KEY(200) THEN bat_y[1]=bat_y[1]-2
  IF KEY(208) THEN bat_y[1]=bat_y[1]+2

  // Bat at upper/lower border?
  IF bat_y[num]<0   THEN bat_y[num]=0
  IF bat_y[num]>416 THEN bat_y[num]=416
 NEXT
```

The appropriate codes for the KEY() command can be found by running the key code tool (under the menu option "tool/key codes").
Now again compile and start the program. With the keys "A, Z" and "up, down" you should be able to control the bats now.

Next, we move the ball based on its current x/y speed. Add this to the MoveAll subroutine :

```
// ball
ball_x=ball_x+ball_sx
ball_y=ball_y+ball_sy
```

Collision with Border

If the ball touches the ceiling or the floor, we simply reverse its y-speed:

```
// ball at bottom
IF ball_y>464
 ball_y=464
 ball_sy= -ball_sy
ENDIF
// ball at ceiling
IF ball_y<0
 ball_y=0
 ball_sy= -ball_sy
ENDIF
```

If the ball hits the left/right edge of the screen, a player scores one point and the ball is reset to the centre again.

```
// ball left border - > score for player 1
IF ball_x<0
 score[1]=score[1]+1
 GOSUB ResetBall
ENDIF
// ball right border - > score for player 0
IF ball_x>624
 score[0]=score[0]+1
 GOSUB ResetBall
ENDIF
```

Collision with Bat

We loop for all players:

```
FOR num=0 TO 1
```

For each player we determine whether the ball is moving towards the bat. By doing this we ensure that the collision of the bat and ball is only checked - and hence processed - once for each bat. Without this, any time the ball hit a bat the ball's direction would change as many times as there were bats on the field.

```
IF (ball_sx<0 AND num = 0) OR (ball_sx>0 AND num=1)
```

If the ball is moving towards the bat being processed then we examine whether the two rectangles for the ball and the bat overlap. If the ball touches a bat, it's x-speed is reversed (i.e. the ball bounces off the bat). The speeds in X and Y both also will increase with each bounce. We make the speeds change unequally as otherwise the angle of the ball's trajectory would never change.

```
col=BOXCOLL(bat_x[num], bat_y[num], 16, 64, ball_x, ball_y, 16, 16)
IF col=TRUE
```

```
// turn ball speed in X direction
ball_sx = -ball_sx
// speed up ball
ball_sx = ball_sx * 1.2
ball_sy = ball_sy * 1.05
```

Closing the IF and FOR statements:

```
   ENDIF
 ENDIF
NEXT
```

Voilà. The game is finished. Before writing your own game, it's important to ensure that you understand what each line of the code is doing.

Challenges

- Check the commands in the reference and try to extend the game from 2 to 4 players (put the bats at the top and bottom of the screen).
- Provide a computer player, so that you can play alone as well. As a hint, you only have to determine, whether the Y-coordinate of the ball is above or below the centre of the computer's bat, and then move the bat accordingly
- Depending on the impact position of the ball on the bat, change the flight angle of the ball.

The program offers lot of freedom for your own ideas. Have fun playing with it.

The entire source code:

```
// Pong

DIM bat_y[2]
DIM bat_x[2]
DIM score[2]

// Voreinstellungen / Setup values
GOSUB Init

// Hauptschleife / Main Loop
main:
 GOSUB MoveAll
 GOSUB ShowAll
GOTO main



// ------------------------------------------------------------ //
// -=#  INIT  #=-
// ------------------------------------------------------------ //
SUB Init:
 GOSUB ResetBall

 // Spielfeld zeichnen, als Hintergrund nutzen
 // Draw playfield, use as background bitmap
 BLACKSCREEN
 DRAWRECT 0,   0, 640, 16,  RGB(255, 255, 255)
 DRAWRECT 0, 464, 640, 16, RGB(255, 255, 255)
 DRAWRECT 312, 0, 16, 480, RGB(255, 255, 255)
 USEASBMP

 // Schläger 0 setzten / Reset Bat 0
 bat_y[0]=240; bat_y[1]=240
```

```
 // Schläger 1 setzten / Reset Bat 1
 bat_x[0]=16; bat_x[1]=600

 // Klassischer Zeichensatz / Classic Font
 LOADFONT "FontBW.bmp", 0
ENDSUB // INIT

// ------------------------------------------------------------ //
// -=#  RESETBALL  #=-
// ------------------------------------------------------------ //
SUB ResetBall:
 // Diese Variablen sind als LOCAL definiert:
 // These variables are defined LOCAL:
 // void
 ball_x=320
 ball_y=240

 IF ball_sx<0
  ball_sx=1
 ELSE
  ball_sx=-1
 ENDIF

 ball_sy=1
ENDSUB // RESETBALL


// ------------------------------------------------------------ //
// -=#  SHOWALL  #=-
// ------------------------------------------------------------ //
SUB ShowAll:
 // Die Schläger anzeigen / Show the bats
 FOR num=0 TO 1
  DRAWRECT bat_x[num], bat_y[num], 16, 64, RGB(255, 255, 255)
  PRINT score[num], num*320 + 32, 16
 NEXT
 // Der Ball / The ball
 DRAWRECT ball_x, ball_y, 16, 16, RGB(255, 255, 255)
 SHOWSCREEN
ENDSUB // SHOWALL


// ------------------------------------------------------------ //
// -=#  MOVEALL  #=-
// ------------------------------------------------------------ //
SUB MoveAll:
// Paddles
 FOR num=0 TO 1
  // Tasten / Keys:  A, Y
  IF KEY(30) THEN bat_y[0]=bat_y[0]-2
  IF KEY(44) THEN bat_y[0]=bat_y[0]+2
  // Tasten / Keys:  /\, \/
  IF KEY(200) THEN bat_y[1]=bat_y[1]-2
  IF KEY(208) THEN bat_y[1]=bat_y[1]+2

  // Schläger am Rand? / Bat at border?
  IF bat_y[num]<0   THEN bat_y[num]=0
  IF bat_y[num]>416 THEN bat_y[num]=416
 NEXT

// Ball
 ball_x=ball_x+ball_sx
 ball_y=ball_y+ball_sy

 // Ball unten am Rand / Ball at lower border
 IF ball_y>464
  ball_y=464
  ball_sy=-ball_sy
 ENDIF

 // Ball oben am Rand / Ball at upper border
 IF ball_y<0
  ball_y=0
  ball_sy=-ball_sy
 ENDIF

 // Ball Linker Rand / Ball at left border
```

```
IF ball_x<0
 score[1]=score[1]+1
 GOSUB ResetBall
ENDIF

// Ball rechter Rand / Ball at right border
IF ball_x>624
 score[0]=score[0]+1
 GOSUB ResetBall
ENDIF

// Pong
FOR num=0 TO 1 // Für jeden Spieler / For each player
 // Bewegt sich der Ball auf den Schläger 'num' zu?
 // Does ball move toward bat 'num'
 IF (ball_sx<0 AND num = 0) OR (ball_sx>0 AND num=1)
  col=BOXCOLL(bat_x[num], bat_y[num], 16, 64,   ball_x, ball_y, 16, 16)

  IF col=TRUE
   // Ballgeschwindigkeit in X umdrehen / Flip ball's X-direction
   ball_sx= -ball_sx
   // Schneller machen / Speed up
   ball_sx = ball_sx * 1.2
   ball_sy = ball_sy * 1.05
  ENDIF
 ENDIF
NEXT
ENDSUB // MOVEALL
```

PART #2

Technical Details of GLBasic

Anatomy of a GLBasic Program

In order to create a GLBasic execute you need at least two files :

- The project file (created by either Project → New in the File menu, or by selecting “Create New Project” in the Project wizard when the program starts, or by loading in an existing project.
- The main file. This is also created when a new project is created. Which main file is used can also be modified by the user to a file that isn't the default one

Optionally, you can also have :

- One or more Source files. These are created by selecting either New file in the Files tab (in Project Management), or by creating a new source file, saving it and then manually adding it, by selecting Add File in the Files tab

Without the first two, an executable can't be created – you cannot compile a file on its own, and a project without any files will not compile.

The projects name should reflect what you want to call the resulting executable – whilst you certainly can rename the application once its been created in Windows, Linux etc., there is no point in continually renaming a file from “test1.exe” to “Mighty Warriors.exe” all the time.

If you do need to change the executable name, it can be done in Project → Options and then change the name of the executable. It would also be a good idea to also rename the project file (ends in .gbas) to the new name too.

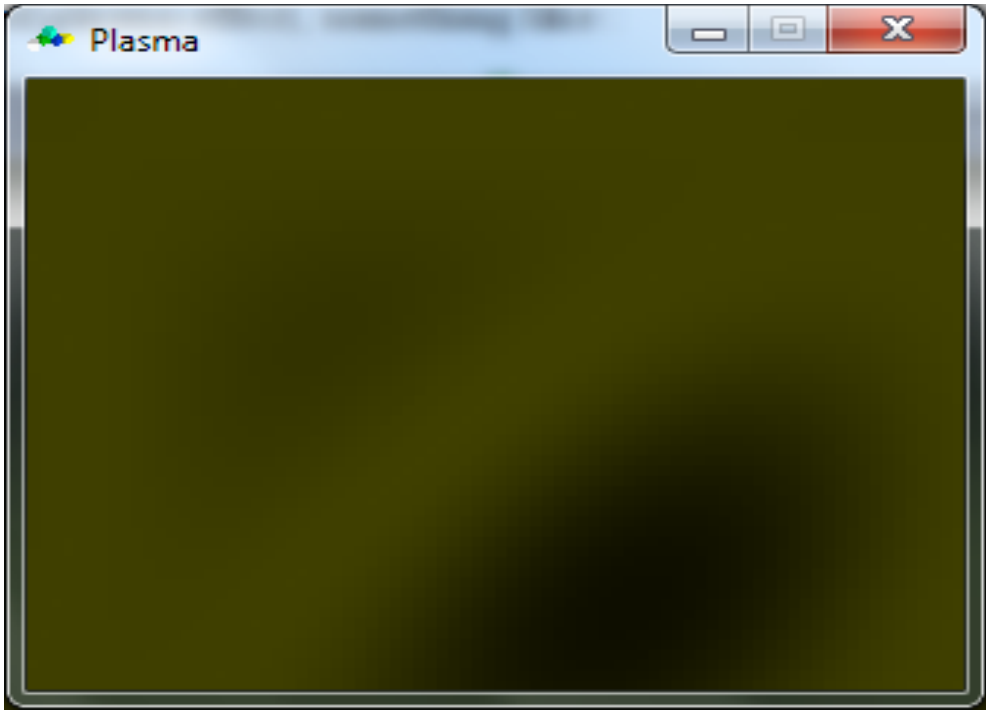
Executable sizes

File sizes for the same program can vary across platforms. Take the following code :

```
// --------------------------------
// Project: Plasma
// Start: Wednesday, March 11, 2009
// IDE Version: 6.184

TYPE tCosTable
	value
ENDTYPE

TYPE tColour
	value%
ENDTYPE

TYPE TPlasma
	cosTable[] AS tCosTable
	colourTable[] AS tColour
	screenWidth%;screenHeight%
	wave%
	stp%

	FUNCTION Initialise_Plasma%:
	LOCAL loop%
	LOCAL cTable AS tCosTable
	LOCAL cColour AS tColour

		DIM self.cosTable[0]
		DIM self.colourTable[0]

		GETSCREENSIZE self.screenWidth%,self.screenHeight%

		//  Create COS table
		FOR loop%=0 TO self.screenWidth%*2
			cTable.value=SIN(360.0*loop%/320.0)*32+32
			DIMPUSH self.cosTable[],cTable
		NEXT

		FOR loop%=0 TO 255
			cColour.value%=RGB(loop%,loop%,0)
			DIMPUSH self.colourTable[],cColour
		NEXT

		self.wave%=0
		self.stp%=14
	ENDFUNCTION

	FUNCTION Update_Plasma%:
	LOCAL x%,y%
	LOCAL pos1,pos2

		INC self.wave%,3
		IF self.wave%>320
			self.wave%=0
		ENDIF

		FOR y%=0 TO self.screenHeight%-1 STEP self.stp%
		pos1 = self.cosTable[y%+self.wave%].value

			FOR x% = 0 TO self.screenWidth%-1 STEP self.stp%
				pos2 = (self.cosTable[x%+self.wave%].value + self.cosTable[x%+y%].value
					 + pos1)
				STARTPOLY -1
					POLYVECTOR x%+self.stp%,y%,0,0,
						self.colourTable[self.cosTable[pos2].value].value
					POLYVECTOR x%,y%,0,0,
						self.colourTable[self.cosTable[pos2].value].value
					POLYVECTOR x%,y%+self.stp%,0,0,
						self.colourTable[self.cosTable[pos2].value].value
					POLYVECTOR x%+self.stp%,y%+self.stp%,0,0,
```

```
                                        self.colourTable[self.cosTable[pos2].value].value
                    ENDPOLY
                NEXT
            NEXT
        ENDFUNCTION
ENDTYPE

LOCAL plasma AS TPlasma

plasma.Initialise_Plasma()
WHILE KEY(1)=0
        plasma.Update_Plasma()
        SHOWSCREEN
WEND
```

This produces an animated plasma effect, something like :



The executable currently created in V9, for some of the supported platforms is as follows :

Platform	Resulting executable size (Debug Mode)	Resulting executable size (Release Mode)
Windows	13,044,524 bytes	13,038,380 bytes
Mac (Universal)	11,755,308 bytes	11,753,772 bytes
Mac (Intel)	10,425,132 bytes	10,414,380 bytes
Linux	729,720 bytes	729,720 bytes
Windows CE	1,431,552 bytes	1,431,552 bytes
GP2X	3,439,868 bytes	3,439,868 bytes
GP2X-WIZ	1,814,924 bytes	1,814,924 bytes
Caanoo	1,571,260 byes	1,571,260 bytes
Pandora	732,644 bytes	732,644 bytes
webOS	821,292 bytes	821,292 bytes
Android	1,265,758 bytes	1,265,755 bytes

As you can see, the executable size can vary in size depending on platform. It should also be noted that even though there are no debug statements in the test code, the Windows executable is less if Release mode is used – a saving of around 6K

If debug statements were added to the above code, the file size between Release and Debug mode would be greater, depending on how many there were.

Note : In versions before V9.052, a Mac application was a Universal binary, meaning an Intel and PowerPC version was created for running on either of these two machines. Since V9.052, in order to allow GLBasic programs to be put into the Mac Application Store, GLBasic can compile as either an Intel-only program or as a Universal which allows a program to be run on either Intel or PowerPC computers.

2D Graphics

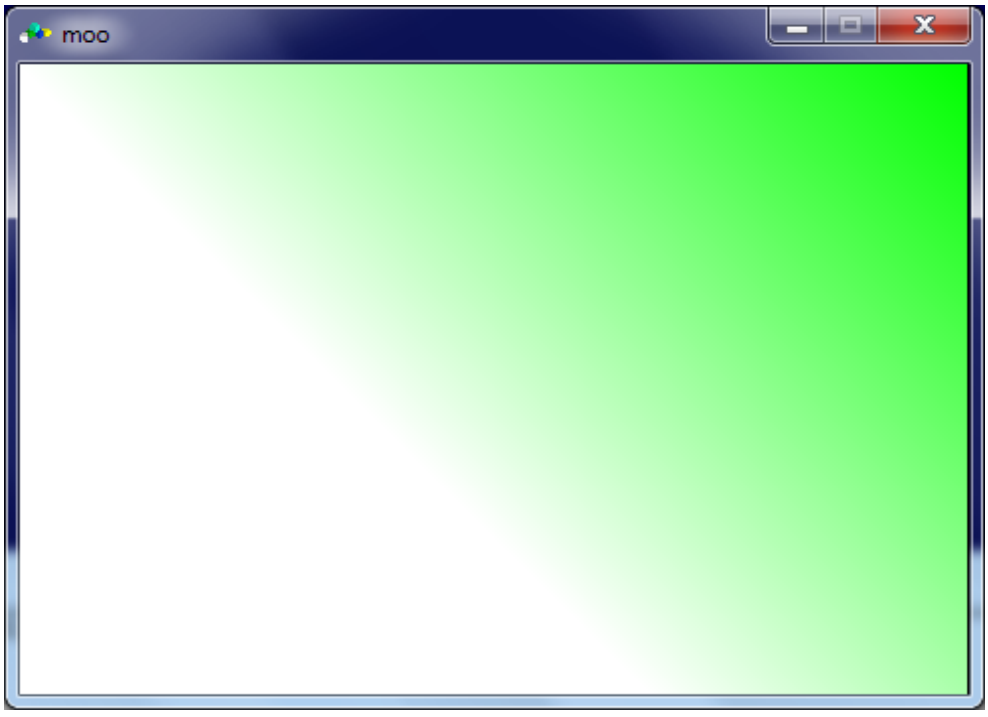
At the moment, GLBasic can load in BMP, PNG and JPEG files to be used as sprites.

PNG files have the advantage of being smaller than BMP files and an alpha mask can be applied to the graphic when being designed, removing the need to define a colour in your program, with the SETTRANSPARENCY command.

There are also other ways to create graphics, from the simplest (by using SETPIXEL to the more powerful (but more complicated) STARTPOLY/POLYVECTOR/POLYNEWSTIP/ENDPOLY set of commands. For example, the following :

```
STARTPOLY 0 // Bitmap = No.0
  POLYVECTOR  0,   0,  0,  0, RGB(255, 255, 255)
  POLYVECTOR   0, 479,  0, 64, RGB (255, 255, 255)
  POLYVECTOR 479, 479, 64, 64, RGB(255, 255, 255)
  POLYVECTOR 479,  0, 64,  0, RGB(  0, 255,   0)
ENDPOLY
SHOWSCREEN
KEYWAIT
```

Produces a gradient filled box :



Text Font

By default, the text font is small and hard to read. This is designed to remind you to change the default font to something more readable!

Up until recently, GLBasic has only been able to print fixed-width text. If you have need proportional text printing, you would have had to use a routine like Moru's system (available at http://gamecorner.110mb.com/index.php?page=bitmap-fonts-proportional).

Update 8.148 has now fixed that inflexibility, with the upgrade of several commands – see the `PRINT` and `LEN` descriptions for more information

Sprites

Sprites are an easy way of displaying loaded graphics. Each sprite command takes an X and Y coordinate, and usually these coordinates are used to draw the sprite from the top-left corner of the sprite. However, depending on the command, these coordinates can define the middle of a sprite, as defined.

Command	How X and Y coordinates are treated
DRAWSPRITE	X and Y draw from top-left coordinates
DRAWANIM	X and Y draw from top-left coordinates
ROTOSPRITE	X and Y draw from top-left coordinates
ROTOZOOMSPRITE	X and Y draw from top-left coordinates, unless scale is different from 1.0
ROTOZOOMANIM	X and Y draw from top-left coordinates, unless scale is different from 1.0
STRETCHSPRITE	X and Y draw from top-left coordinates
STRETCHANIM	X and Y draw from top-left coordinates
ZOOMSPRITE	X and Y draw from top-left coordinates, unless scale is different from 1.0

3D Graphics

It should be noted that GLBasic is no slouch in the 3D department either, capable of using shaders (if available for the platform) and drawing lines and dots, complete with collision if needed.

In Debug mode, the collision sphere (or box) will be visible and will change colour if there is a collision

No collision sphere in Release Mode :

Showing the collision sphere in Debug Mode :

Incorrectly using the 3D command may produce just a blank screen or display objects just once. The correct order of using the commands is (for each player) :

Set viewport (using `VIEWPORT`)
`X_MAKE3D`
`X_CAMERA`
`X_CULLMODE` (if needed)
Lighting (`X_SPOT_LT` if needed) (except when skybox/skysphere is used)
Fog (`X_FOG`) if needed
[*Display skybox/skysphere followed by lighting and then `X_CLEAR_Z`*]
Display all solid 3D objects
Display all ghosted 3D objects (must be rendered last)
2D graphics (`X_MAKE2D`) if needed

Graphics In General

It should be noted, that, unlike some other game making languages, GLBasic requires all graphics to be rendered each frame. Whilst this may seem cumbersome, done in an efficient manner (so that all sprites and 3D objects can be rendered in a loop), there would be little loss of game speed.

An important note should be made about `SMOOTHSHADING`. By default, this will anti-alias 2D graphics, which can cause side-effects when scaling or rotating a sprite. You may also get a red outline if `SMOOTHSHADING FALSE` isn't used.

Sounds And Music

All platforms use WAV formats for sounds – PCM format only. If a different type of WAV file is loaded, the sound wont be heard.

There are no commands to enable a sound to be repeated (unlike music). However, you can either continually check to see if a sound is playing or, if you know exactly how long the sound is, you could play the sound when a certain amount of time has passed.

Music playing can be a bit tricky depending on platform – for example, generally Windows cant play OGG files, whilst the Mac can and Linux can play both.

It is best to use the most common format for the platform you are targeting, so the user doesn't have to install any extra codecs for playing unfamiliar sound formats.

So, the files allowed are as follows :

Platform	Music type allowed
Windows	MP3*
Mac	OGG, AIF and CAF*
Linux	MP3 or OGG
GP2X	MP3 (poorly) or OGG
GP2X-WIZ	MP3 (poorly) or OGG
Windows CE	MP3
iPhone	OGG
Pandora	OGG
XBox	OGG
Palm Pre	MP3
Android	OGG

- Unless additional codecs are installed to support the alternative format.#

Sounds and music use a mixture of OpenAL, DirectSound and SDL_mixer (depending on platform).

There may also be a limit (especially for mobile devices) on the size of a WAV that can be loaded.

File I/O

With the latest version of GLBasic, the LOAD commands (`LOADSPRITE`, `LOADSOUND` etc.) will now pass the value “0003 no such file” to the `GETLASTERROR$` command.

Using C/C++ (or perhaps another language)

Using another language in GLBasic is quite simple, with a myriad of choices. Usually C/C++ is the language of choice and thus will be covered here.

There are many ways of including C/C++ source code, or with a DLL, dylib or .a file : There is REQUIRE, DECLARE (and its variants) as well as DLLCALL

However, in most cases, INLINE/ENDINLINE is used.

When doing so, all C/C++ should be put in a source file (and not the main file as compiler errors may occur).

The layout of the main INLINE block (which is usually used to define variables, types etc.) should look like :

```
INLINE
	}

	// Add classes, #include variable and type definitions, extern calls here

	namespace __GLBASIC__ {
ENDINLINE
```

All standard C library routines are available, so its possible to use things like floor or ceil :

```
INLINE
	}

	// Add classes, #include variable and type definitions, extern calls here
	extern "C" double floor(double x);

	namespace __GLBASIC__ {
ENDINLINE

FUNCTION floorvalue:value
INLINE
	return DGInt(floor(value));
ENDINLINE
ENDFUNCTION
```

C source code cant be called directly (unlike DLL's), so it always has to be wrapped into a function. This does mean however, that INLINE can't be used with extended types.

Arrays (DGArray) in C using GLBasic commands are quite easy. Essentially, you need to define a class holding all the variables needed (much like a TYPE). These variables should be defined in the public section, otherwise you may not be able to put values into a class in order to push into the array.

An example of this is :

```
INLINE
	}
```

```
	// Add classes, #include variable and type definitions, extern calls here
	extern "C" double floor(double x);

	class anything {
	public :
		int x;
	};

	DGArray<anything> anythingArray;

	namespace __GLBASIC__ {
ENDINLINE
FUNCTION arrayTest%:
INLINE
	anything temp;

		temp.x=5;
		DIM(anythingArray,0);
		DIMPUSH(anythingArray,temp);
		DEBUG(DGStr("Array size : ")+DGStr(BOUNDS(anythingArray,0))+DGStr("\n"));
		DEBUG(DGStr("Value of index 0 :")+anythingArray(0).x+DGStr("\n"));
ENDINLINE
ENDFUNCTION
```

This examples calls GLBasic commands within C to push data into an array (`DIM` to clear our array, `DIMPUSH` to push the class into the array and `BOUNDS` to check its size).

DGStr is also used to output text to the Debug window.

There are various types which should be used to maintain compatibility and data integrity between GLBasic code and C/C++ code :

Type	C Equivalent
DGInt	Floating point number, the size depending on processor : iPhone/ARM Processor : float (4 bytes) Other Processors : double (8 bytes)
DGNat	Signed 32-bit number
DGStr	A class for string manipulation (somewhat like Microsoft's CString class), and includes the following functions : len – Length of a string c_str – Return a char * to the text GetStrData(num) – Get a char * to **num** characters
DGIntArray	Array of DGInt
DGNatArray	Array of DGNat
DGArray	Array of DGStr

Accessing arrays is done using () brackets (and not the usual [] brackets).

Dummy Function

A dummy function (a function with nothing in it) may be needed after the main program, if your INLINE code goes out of the GLBasic namespace.

Specific C functions

There are two specific functions available :

Function	Description
`GLBASIC_HWND()`	Gives the HWND (handle) of the GLBasic main window
`GETSUBADDRESS(const DGStr& name)`	Returns a pointer to a `SUB` - analogous to `CALLBYNAME`. A `SUB` is of type: DGInt(*sub)();

Platform headers are some platforms are available at the GLBasic website (http://www.glbasic.com/files/headers.rar) , although they may be out of date.

Inline Preprocessor Variables

When using `INLINE`, various pre-processor variables are available

Preprocessor	Meaning
GLB_DEBUG	Compiling in debug mode (if present)
GLB_VERSION	GLBasic version
GLB_CONSOLE	Compiling in Console mode (if present)
WIN32	Compiling for Windows
WIN32WCE (or WINCE)	Compiling for Windows CE
WIN32DLL	Compiling as a Windows DLL
GP2X	Compiling for GP2X
GP2XWIZ	Compiling for WIZ
CAANOO	Compiling for GPH Caanoo
XBOXLINUX	Compiling for Xbox Linux
PANDORA	Compiling for Pandora
OSXUNI	Compiling for Mac OSX
LINUX	Compiling for Linux
IPHONE	Compiling for iPhone/iPod/iPad
WEBOS	Compiling for the Palm Pre
PALMPIXI	Compiling for the Palm Pixi
ANDROID	Compiling for an Android machine

External Libraries

You can use external static libs with the -lnk line in Platform → Options. Just put the libXXX.a file in the project's directory and type int that edit box: "-lXXX". You can then call the function with IMPORT as if it was a built-in function.

Useful Code

The GLBasic forums contain many useful C routines including :

Allocate Memory

This routine allows you to allocate a block of memory, and to read and write values (bytes, words, strings to the allocated memory).

```
// ------------------------------- //
// Project: TestMalloc
// Start: Monday, March 02, 2009
// IDE Version: 6.174

FUNCTION dummy:
ENDFUNCTION

INLINE
	}

	typedef unsigned int size_t;

	extern "C" void *memcpy(void *destination, const void *source, size_t num );
	extern "C" void *memset(void *ptr, int value, size_t num);

	template <typename type, typename val> void __poke(DWORD Addr, val Value)
	{
		*((type*)(Addr))=(type)(Value);
	}

	template <typename type> type __peek(DWORD Addr)
	{
		return *((type *)(Addr));
	}

namespace __GLBASIC__ {
ENDINLINE

FUNCTION AllocMem%:size%
INLINE
	return (DGNat) new char[size];
ENDINLINE
ENDFUNCTION

FUNCTION FreeMem%:mem%
INLINE
char *temp;

	temp=(char *) mem;
	delete[] temp;
ENDINLINE
ENDFUNCTION

FUNCTION ZeroMemory:mem%,size%
INLINE
	memset((char *) mem,(char) 0,size);
ENDINLINE
ENDFUNCTION

FUNCTION FillMemory:mem%,size%,val%
```

```
INLINE
	memset((char *) mem,(char) val,size);
ENDINLINE
ENDFUNCTION

FUNCTION CopyMemory%:store%,read%,size%=-1
INLINE
	char value;

	IF (size<0)
	{
		value=__peek<char>((DWORD) READ);
		WHILE (value!=0)
		{
			__poke<char>((DWORD) store,value);
			READ++;
			store++;
		}

		RETURN TRUE;
	}
	ELSE
	IF (size>0)
	{
		memcpy((char *) store,(char *) READ,size);
		RETURN TRUE;
	}
	ELSE
	{
		RETURN FALSE;
	}
ENDINLINE
ENDFUNCTION

FUNCTION pokeB:mem%,val%
INLINE
	__poke<char>((DWORD) mem,char(val));
ENDINLINE
ENDFUNCTION

FUNCTION pokeS:mem%,val%
INLINE
	__poke<short>((DWORD) mem,short(val));
ENDINLINE
ENDFUNCTION

FUNCTION pokeL:mem%,val%
INLINE
	__poke<int>((DWORD) mem,val);
ENDINLINE
ENDFUNCTION

FUNCTION pokeF:mem%,val
INLINE
	__poke<float>((DWORD) mem,(float) val);
ENDINLINE
ENDFUNCTION

FUNCTION pokeDF:mem%,val
INLINE
	__poke<double>((DWORD) mem,val);
ENDINLINE
ENDFUNCTION

FUNCTION pokeString:mem%,str$
INLINE
	memcpy((char *) mem,str_Str.c_str(),str_Str.len());
ENDINLINE
ENDFUNCTION

//
-----------------------------------------------------------------------------------------------------
----------------------------
FUNCTION peekB%:mem%
INLINE
	return ((DGNat) (__peek<char>(mem)));
ENDINLINE
```

```
ENDFUNCTION

FUNCTION peekS%:mem%
INLINE
	return ((DGNat) (__peek<short>(mem)));
ENDINLINE
ENDFUNCTION

FUNCTION peekL%:mem%
INLINE
	return ((DGNat) (__peek<int>(mem)));
ENDINLINE
ENDFUNCTION

FUNCTION peekF:mem%
INLINE
	return ((DGInt) (__peek<float>(mem)));
ENDINLINE
ENDFUNCTION

FUNCTION peekDF:mem%
INLINE
	return ((DGInt) (__peek<double>(mem)));
ENDINLINE
ENDFUNCTION

FUNCTION peekString$:mem%,length%=0
LOCAL temp$
LOCAL l%
LOCAL p%

	temp$=""

	IF length%>0
		FOR l%=0 TO length%-1
			temp$=temp$+CHR$(peekB(mem%+l%))
		NEXT
	ELSE
		p%=peekB(mem%)
		WHILE p%<>0
			temp$=temp$+CHR$(p%)
			INC mem%,1
			p%=peekB(mem%)
		WEND
	ENDIF

	RETURN temp$
ENDFUNCTION
```

CRC32

This code will calculate a CRC32 checksum for the contents of a file :

```
// -------------------------------- //
// Project: CRC32
// Start: Monday, August 10, 2009
// IDE Version: 7.078

//! This calculates a CRC-32 value for a file
//\param fileName$ - This is the filename that you want to calculate a CRC value for
//\return - A signed 32-bit CRC value
FUNCTION calculateCRC32%:fileName$
LOCAL channel% =	1
LOCAL bufferSize%	=	1024
LOCAL d%[];	DIM d%[bufferSize%]
LOCAL amount%
LOCAL crc%
LOCAL loop%
LOCAL length

	crc%=0
	IF DOESFILEEXIST(fileName$)=TRUE
		length=GETFILESIZE(fileName$)
```

```
		IF OPENFILE(channel%,fileName$,1)
			DEBUG "length : "+length+"\n"

			// Read in bytes
			FILESEEK channel%,0,0
			amount%=readInBytes(channel%,d%[],bufferSize%,length)
			WHILE amount%>0
				FOR loop%=0 TO amount%-1
					crc%=calcCRC32(d%[loop%],crc%)
				NEXT

				amount%=readInBytes(channel%,d%[],bufferSize%,length)
				DEBUG "."
			WEND

			crc%=calcCRC32End(crc%)
			CLOSEFILE channel%
		ELSE
			DEBUG "Error : "+GETLASTERROR$()+"\n"
		ENDIF
	ENDIF

	RETURN crc%
ENDFUNCTION

@FUNCTION readInBytes%:handle%,d%[],size%,length
LOCAL loop%
LOCAL pos
LOCAL diff

	loop%=0
	WHILE ENDOFFILE(handle%)=FALSE AND loop%<size%
		pos=FILEPOSITION(handle%)
		diff=length-pos
		SELECT diff
			CASE	1
					READUBYTE handle%,d%[loop%]
			CASE	2
					READUWORD handle%,d%[loop%]
			CASE	3
					READUBYTE handle%,d%[loop%]
					INC loop%,1
					IF loop%<size% AND ENDOFFILE(handle%)=FALSE THEN READUWORD
handle%,d%[loop%]
			CASE	>3
					READULONG handle%,d%[loop%]
		ENDSELECT
		INC loop%,1
	WEND

	RETURN loop%
ENDFUNCTION

INLINE
const unsigned long CRC[]={
		0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
		0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
		0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
		0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
		0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
		0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
		0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
		0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
		0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
		0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
		0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
		0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
		0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
		0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
		0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
		0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
		0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
		0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
		0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
		0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
		0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
		0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
```

```
                0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
                0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
                0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
                0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
                0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
                0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
                0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
                0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
                0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
                0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
                0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
                0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
                0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
                0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
                0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
                0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
                0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
                0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
                0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
                0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
                0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
                };
ENDINLINE

@FUNCTION calcCRC32%:value%,crc%
INLINE
                return (CRC[(unsigned char) (crc^(unsigned long) value)]^((crc<<8) & 0x00FFFFFF));
ENDINLINE
ENDFUNCTION

@FUNCTION calcCRC32End%:crc%
INLINE
        return (crc^0xFFFFFFFF);
ENDINLINE
ENDFUNCTION
```

More routines can be found on the GLBasic website

Optional Subroutines

In V7, two optional subroutines were introduced :

Subroutine Name	Details
GLB_ON_QUIT	Called just before the executable finishes Note : • In debugger mode, if this subroutine is not present, the compiler generates CALLBYNAME "GLB_ON_QUIT" *** failed *** message - this can be ignored (V7 only) • Mac programs will crash if any attempt to use any graphic or sound system is made.
GLB_WIN32_MSG	This subroutine is called when an SDL message is available Note : • At the moment, it appears that messages can only be received using C • This subroutine can be called even on non-Windows machines

With version 8 two more were added to take into account the fact that OS 4.x can now multi-task :

Subroutine Name	Details
GLB_ON_PAUSE	This subroutine is called when the running program is minimised or loses focus. Your main loop will still be running, so this subroutine should just set a flag to tell it to pause. No infinite loops, music or graphics commands should be used here Notes : • Only available from Version 8 onwards • Currently only available using release mode • Currently only called on iPhone/iPad and Windows machines. For all other platforms, this routine is not called
GLB_ON_RESUME	This subroutine is called when the running program has its focus returned. Your main loop will still be running, so this subroutine should just unset a flag to tell it to continue. No infinite loops, music or graphics commands should be used here Notes : • Only available from Version 8 onwards • Currently only available using release mode • Currently only called on iPhone/iPad and Windows machines. For all other platforms, this routine is not called This subroutine is controlled by the AUTOPAUSE command

Variables

This lists the different types of variables that are available :

Character at end of variable name :	Denotes :
$	A string variable. Each character takes up one byte. There is no maximum size - you could keep adding characters until you run out of available memory.
# (or nothing)	Floating-point value. On "big" systems, GLBasic's default storage for numbers uses 8 bytes (64 bits), whilst on other systems it is 4 bytes (32 bits).
%	Signed integer with a 4 byte (32 bit) accuracy.

Be aware that values typed directly into your code that are written as integers (whole numbers) may force calculations to be done as integer calculations. This may lead to unexpected results when you assign the result of a calculation to a variable that supports a decimal place but get an integer result back. For example

```
MyNumber% = 1234              // Integer
Result# = MyNumber / 1000     // Result is a float value
```

As "Result" has been defined as a floating point number (that is, one that supports decimal places - indicated by the "#"), you would expect that "Result" would contain the value "1.234". Instead, the compiler determines that the "1000" is an integer value (as it doesn't contain a decimal point) and does an integer based calculation. The result of 1234 / 1000 as an integer calculation is "1".
To force calculations to be done as a floating point calculation, add a ".0" to the end of any number typed into your code that you wish to be interpreted as a floating point number. If you change the above code to be

```
MyNumber% = 1234              // Integer
Result# = MyNumber / 1000.0   // Result is a float value
```

Result will end up with the expected value of "1.234"

Escape Codes

Having a “\” character on its own in a string can cause the compiler to generate a syntax error message. This is because C style escape codes can be used (which begin with a “\”), and thus the next character is regarded as part of the sequence. If there is none, or it is invalid, the aforementioned error will be displayed.

Naming

Variables cannot have the same name as an existing command (you would usually get a compiler error if you try and do so), or existing variable or function. Nor should they be all capital letters, so, for example :

```
CONSTANT LOCALVARIABLE% = 1
```

will generate a compiler error (due `LOCAL` being a command in GLBasic)

Pre-defined constants :

Name	Value
TRUE	1
FALSE	0

With version 8, C/C++ data types can be used for most operations., with the exception of using +, -, * and / and various other commands

Available types are :

```
char            short               int
long            double
```

Line Separators

A "_" character can be used to split parameters and groups of commands over multiple lines, for example :

```
FUNCTION test%:a%,b%,c%, _
                      d%,e%
…
ENDFUNCTION
```

Getting project name

V9.040 allows a function to be called if you need to get the name of the project. This needs to be called within an `INLINE`/`ENDINLINE` pair :

```
FUNCTION returnProgramName$:
INLINE
        return DGStr(__glb_AppName());
ENDINLINE
ENDFUNCTION
```

Compiling

Compiling for platforms just usually consists of either pressing F5 to compile, or selecting the platform from the Compiler → Build Multi-platform option.

However, the iPhone/iPad requires an additional step : Once GLBasic has created the iPhone/iPad project, you need to transfer the resulting XCode project to a Mac, and which point you need to compile on Xcode, making sure the appropriate platform is selected.

Unfortunately, the resulting program cant be run (or even compiled properly) at this stage, as Apple have seem fit make the whole process as complicated as possible (especially if you want to submit your application to the AppStore).

In order to develop for these machines, you need :

- Apple Development Connection account
- Create a provisioning profile (usually takes a day or so)
- Set-up your iPod/iPhone/iPad
- Make sure you have the latest version of XCode for your Mac
- Follow all the directions for putting the license on your Mac
- Compile and hopefully you'll get a running program

You may find that XCode complains about a missing SDK. If so, you need to make sure you have the latest available version.

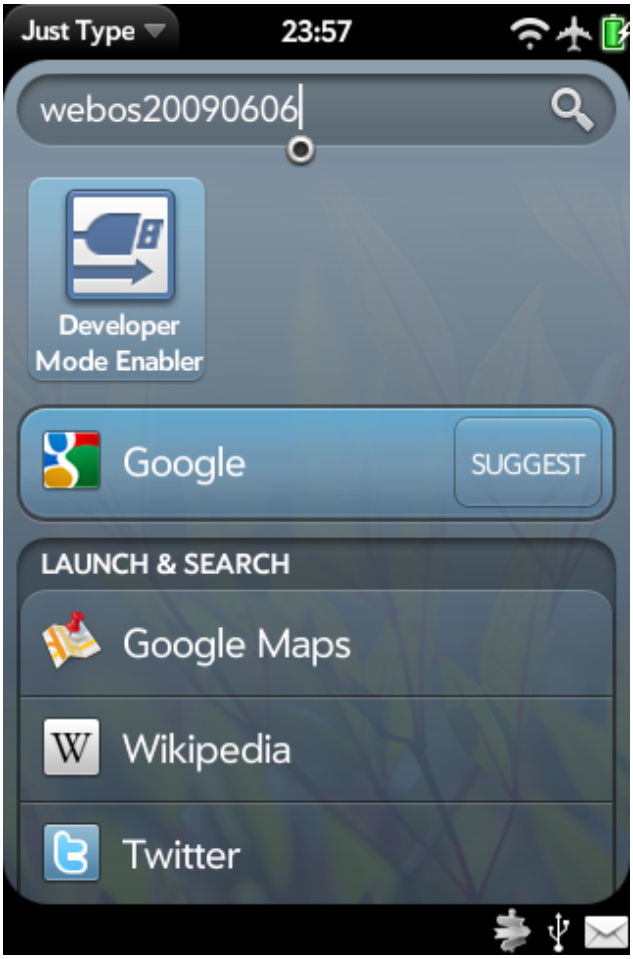
Compiling for Palm (WebOS)

GLBasic ships with everything you need for developing with WebOS devices. The compiling and testing (or running) process goes as follows :

Preparing the device

In card view (with no cards open) type "webos20090606" (without the quotes) in the search box.

There will be an icon displayed called "Developer Mode Enabler". Tap that, and turn developer mode on.



Driver and connection

In the GLBasic program directory, there is a directory called "*compiler\platform\webos\tool\bin\novaterm\drivers*", in which there are two drivers – one for 32 bit systems and one for 64 bit systems. You will need to install the correct version.

It is best to install the drivers **before** connecting the machine, as otherwise one driver installation may fail (there are 4 installed, in total).

There is also a "*pdk-tcprelay-install.bat*" batch file. You have to start this file with administrator rights. When typing "i" for [i]nstall, the service "Palm TCP Relay" is installed, which is required for getting a device shell. If you only want to upload to the device and test, this step might not be necessary for you.

Now connect the device via USB. On the device select "Only Charge" - don't pick "Use as USB drive" !

At this point, you should double click the file "*pdk-device-install.bat*". The program should copy the required files to your device at this point. If there is an error, please make a post in the GLBasic forums.

Programming

Once you have compiled a program in GLBasic for WebOS, a directory "glbwebos" will be created in the .app directory. In the directory will be a file called "appinfo.json" which will need modifying with Notepad. The contents are important, and should be :

```
{
   "id": "com.glbasic.appname_lowercase",
   "version": "1.0.0",
   "vendor": "Your Company Name",
   "type": "pdk",
   "main": "glbasic-webos-exe",
   "title": "WebOS Icon Text",
   "icon": "icon.png"
}
```

The changes will not be overwritten the next time you build.

The directory "Media" is created, but does not contain any files at this time Please copy the files from .app/Media into that folder.

The icon should be a 48 x 48 file with transparency, and is called "icon.png". It is just a matter of replacing the default icon with yours. A complete build and then upload is required to make sure the icon is integrated.

Upload and test

The folder "glbwebos" in the project directory must now be packed and uploaded to the device. That is done with the command (starting from the GLBasic program directory) :

```
"..\GLBasic\compiler\platform\webos\tool\bin\glb_pack_and_install.bat" "...\glbwebos"
```

NOTE : You may need to install the Java Development Environment to get the packing routine running as it uses a bit of Java.

For rapid deployment you can use an GLBasic Editor macro to create a package and upload to the device :

```
CALL "%GLB_MAIN_PATH%\compiler\platform\webos\tool\bin\glb_pack_and_install.bat" "%GLB_PROJ_PATH%\
%GLB_PROJ_NAME%.app"
```

After a few seconds the script should have created a file along the lines of “com-company-appname-version.ipk” and installed it on the device.

This ipk file is the very same file you need for the App Catalog upload.

Debugging

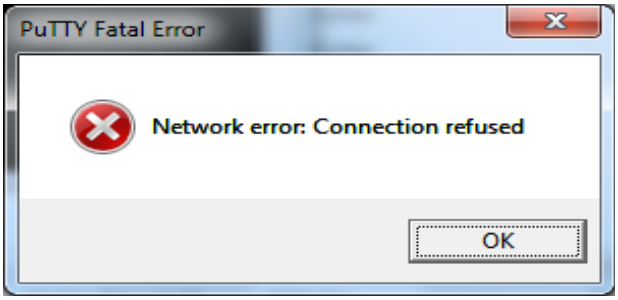
In the directory "GLBasic\Compiler\platform\webos\tool\bin" there's a file "DeviceShell.bat", that opens a command prompt on the device.

Beware, the shell you have has "root" access. That means you have full control over the machine, and any incorrect commands could result in an expensive paper-weight!

You start a program like:

```
# cd /media/cryptofs/apps/usr/palm/applications/
# ls
com.your_company.appname
...
# cd com.your_company.appname
# ./glbasic-webos-exe
...
```

Some platforms may not allow command prompt access to the device, returning the following error message :



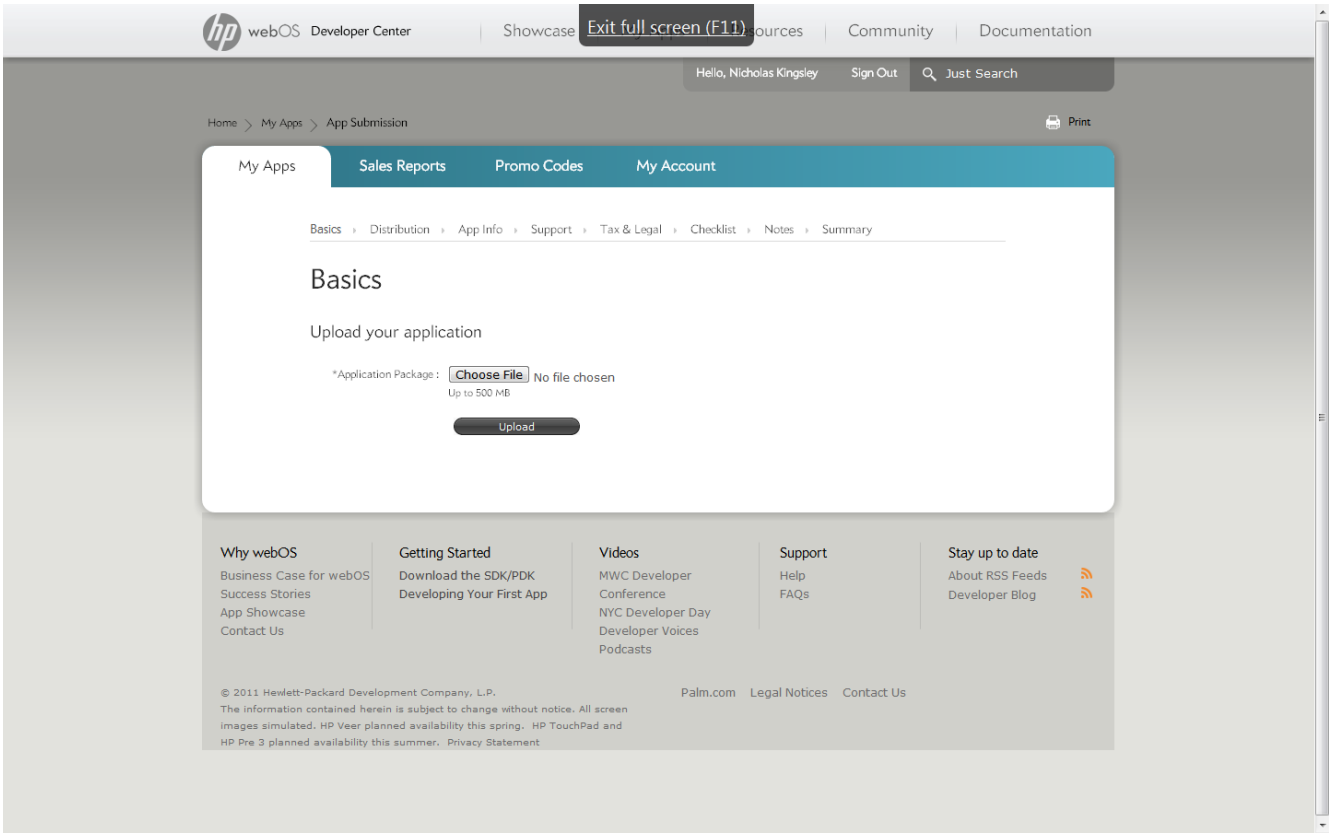
In which case running Novacom or the PDK relay drivers (found in *compiler\platform\webos\tool\bin\novaterm\drivers* should work).

webOS Submission Processes

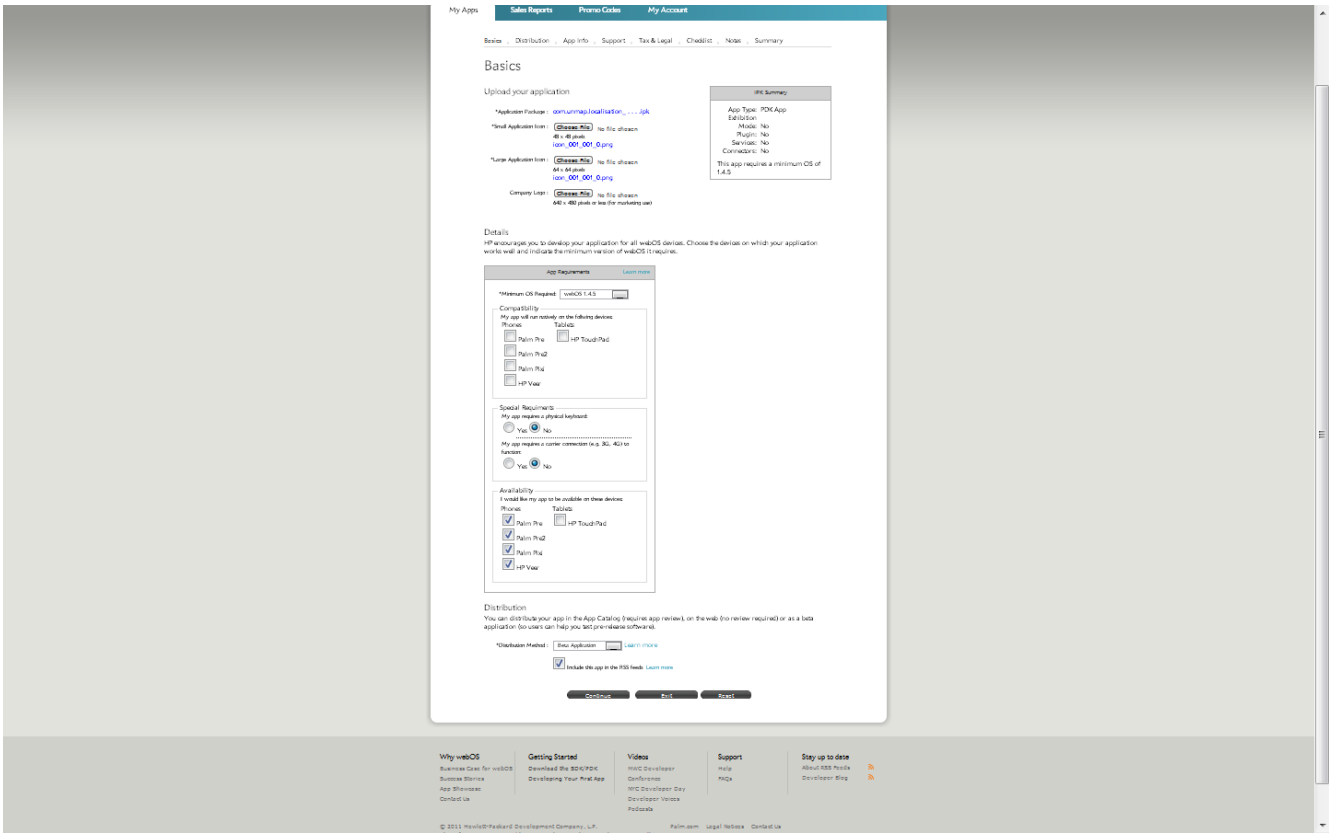
Palm has two somewhat dis-similar submission process. One is for beta applications, where a program is only available through a beta Application Store (or if someone searches for it), and the other is for supplying applications for general release.

Beta applications submission

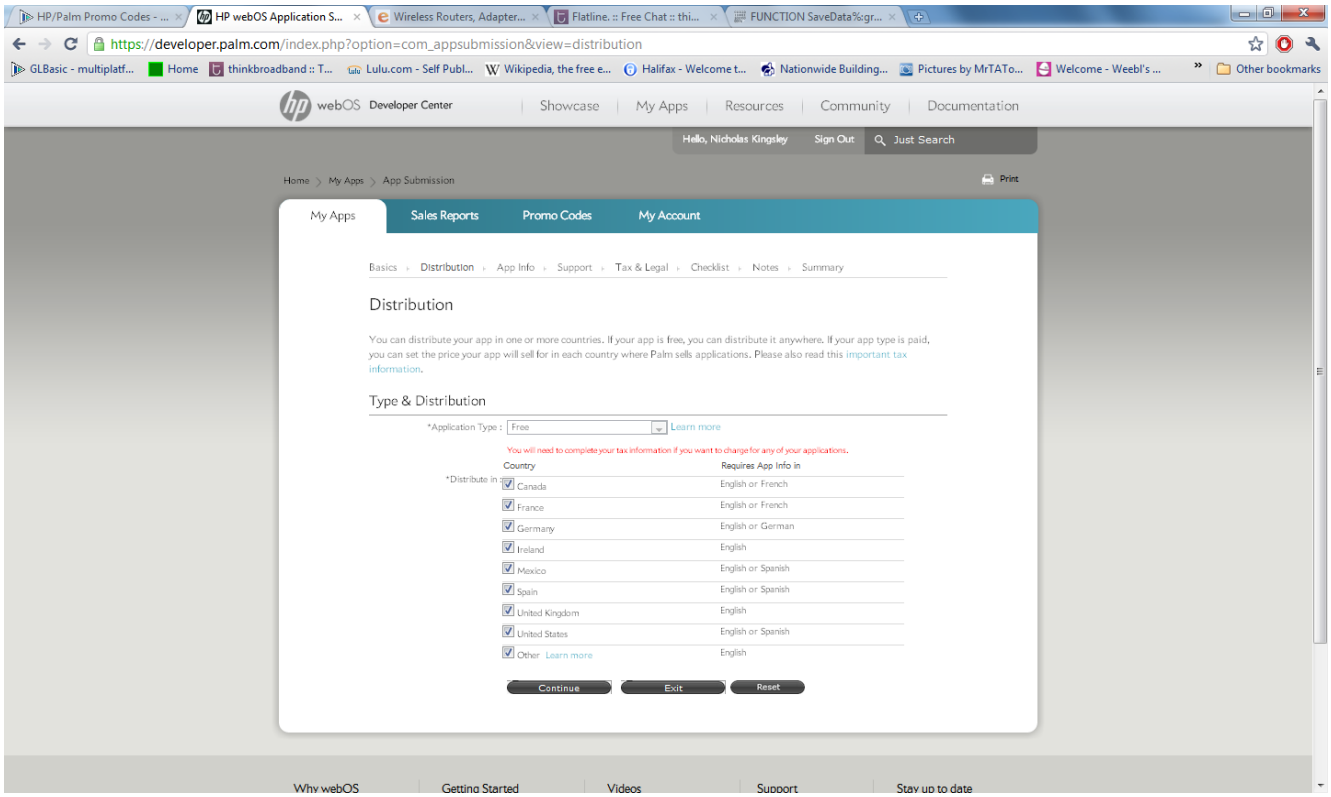
This submission process is quite simple (screen layout may change if Palm upgrades the process) :



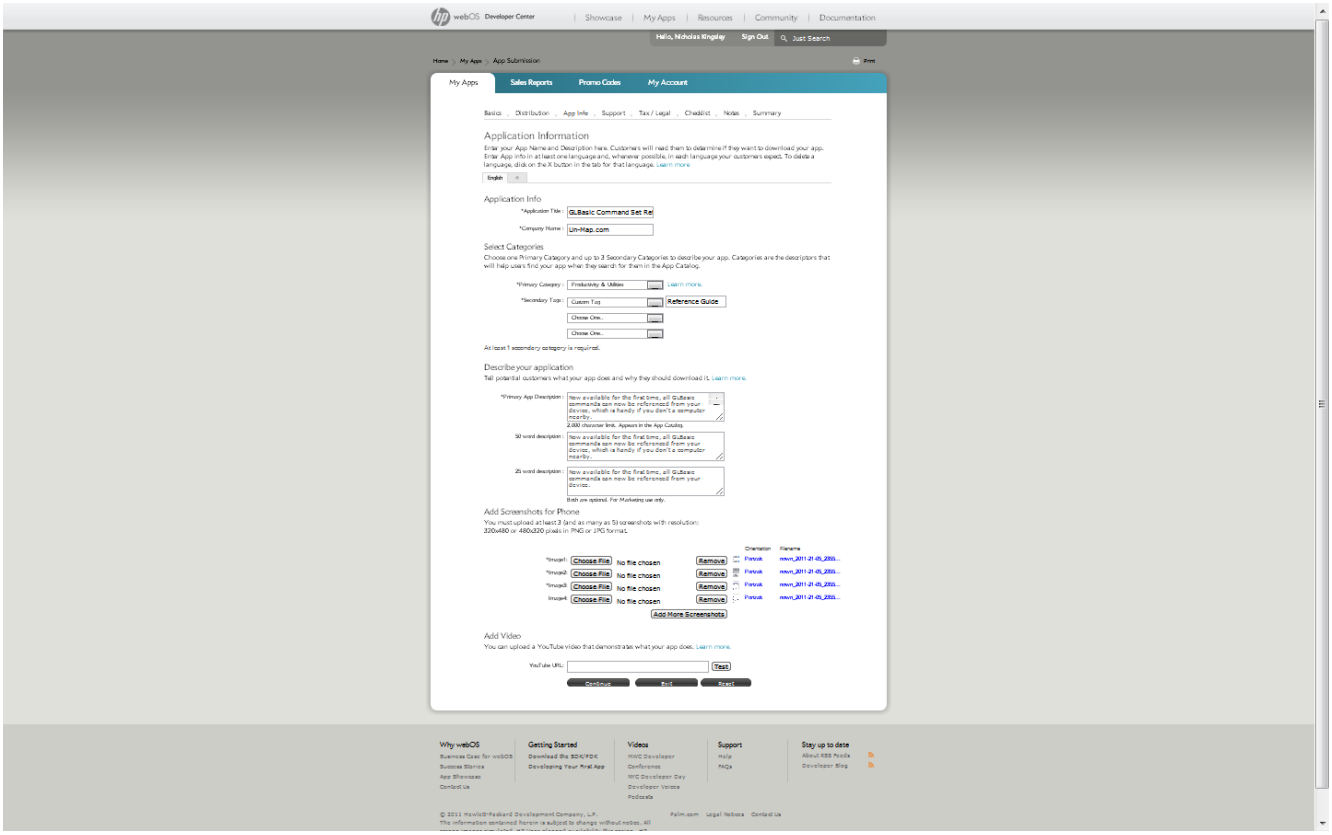
This is the first screen, and it here that you need to upload your IPK file. If the application is already present, then the version must be later than the original for it to be accepted



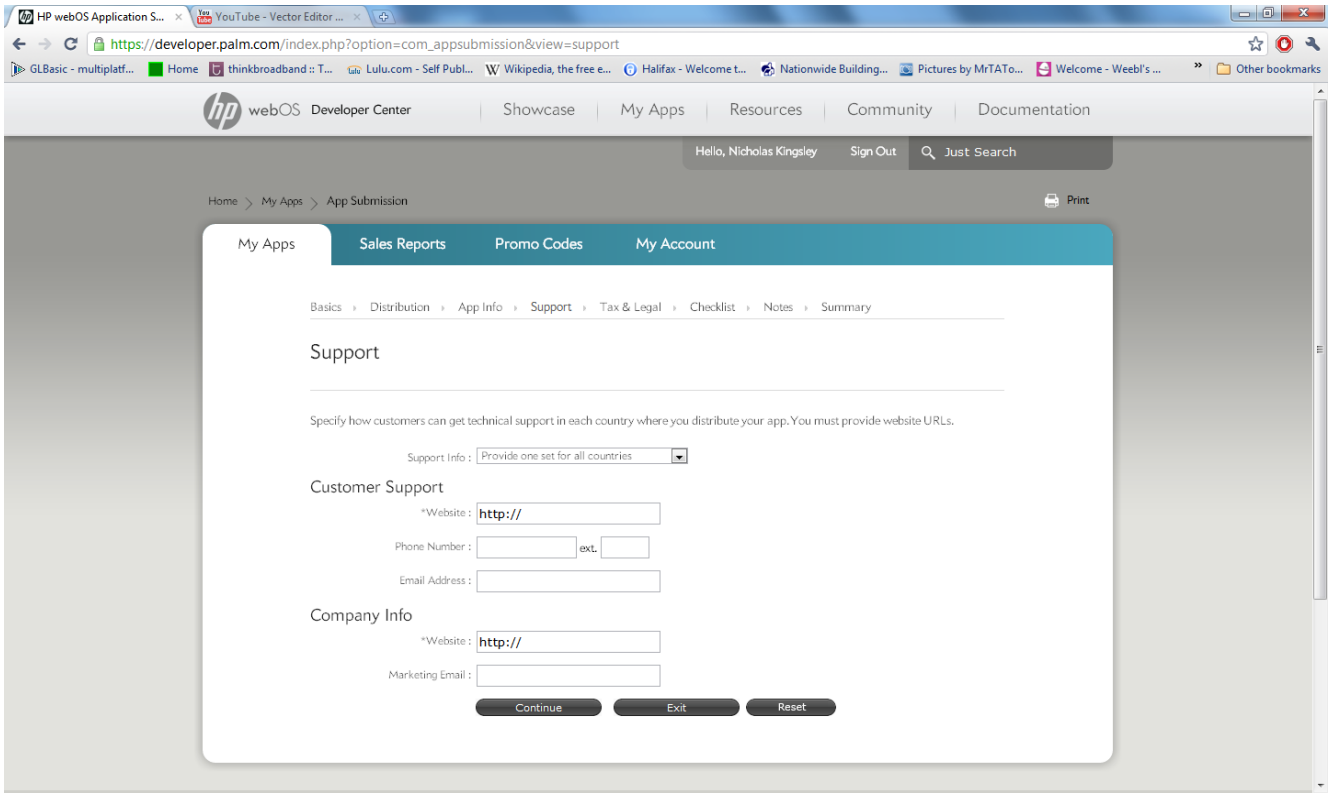
Here you upload your application icons, and select the devices that your application will run on



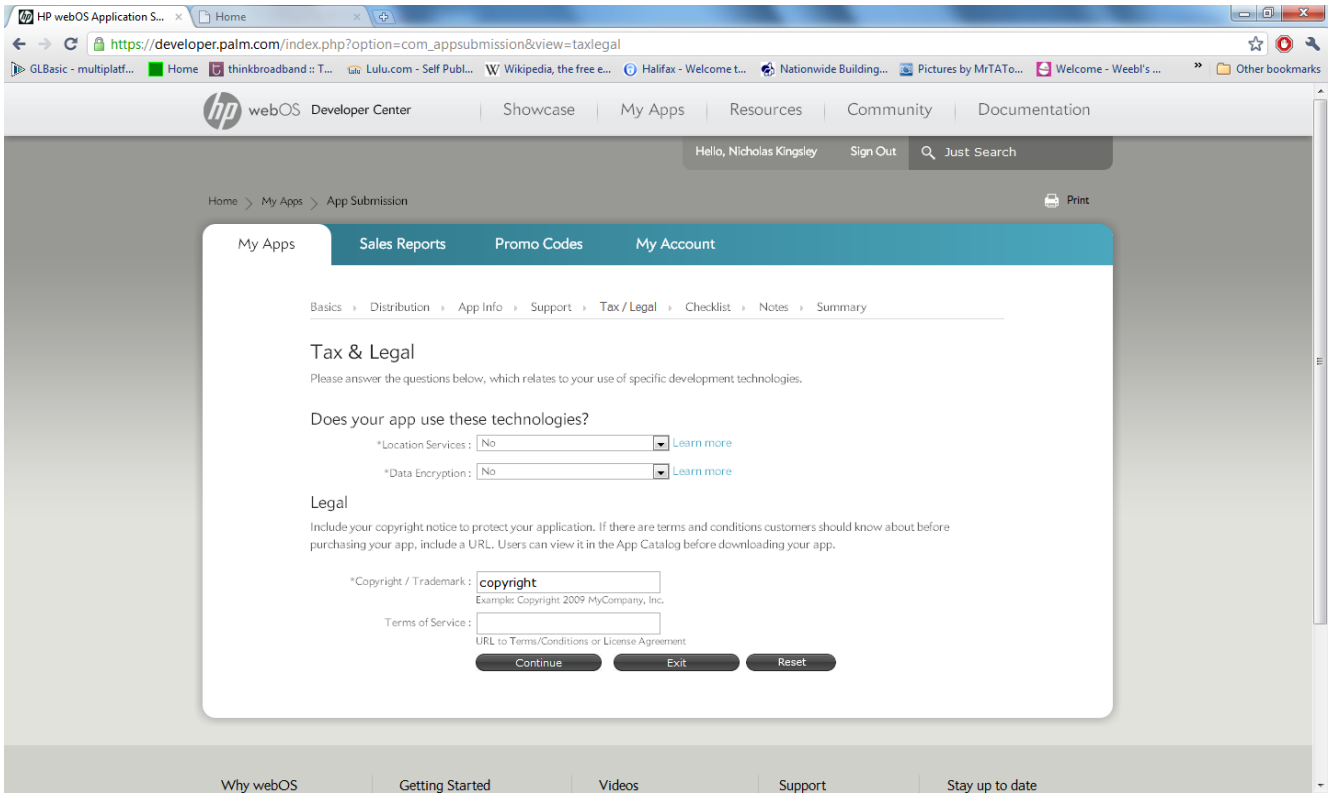
Here you select whether your application is free or paid for (not applicable for Beta applications), and for what countries you want your application to be distributed in.



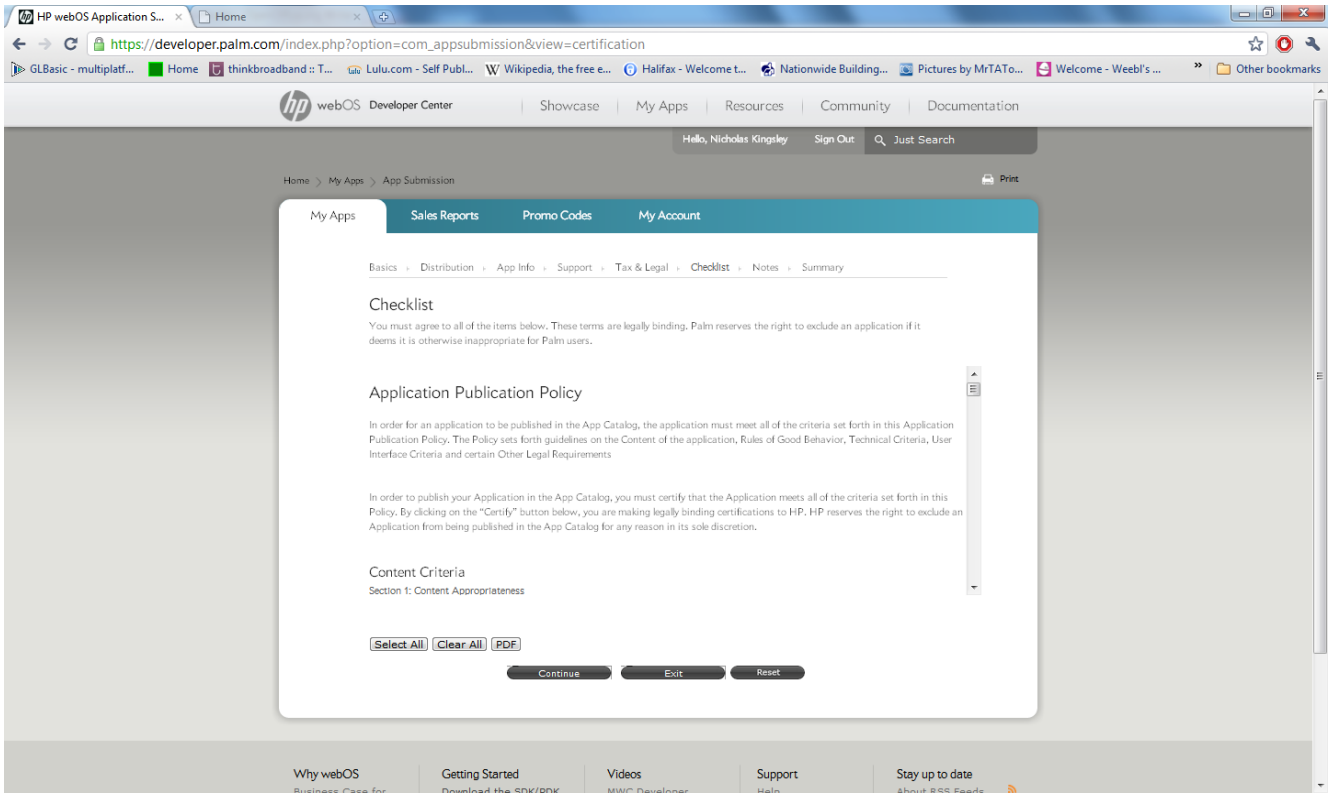
This is where you need to enter in your application details, and to which category it should go in.



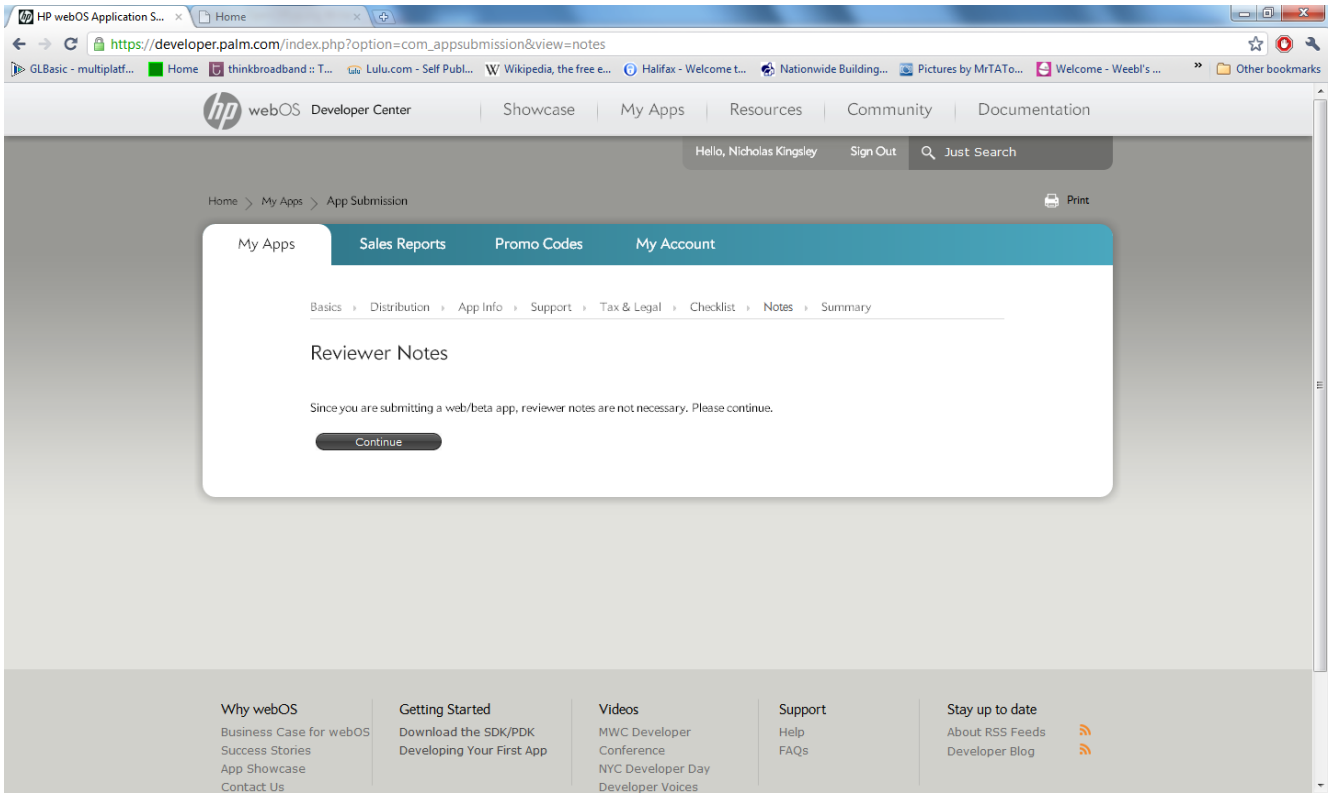
This section details how you should be contacted if a user finds a bug or has a suggestion.



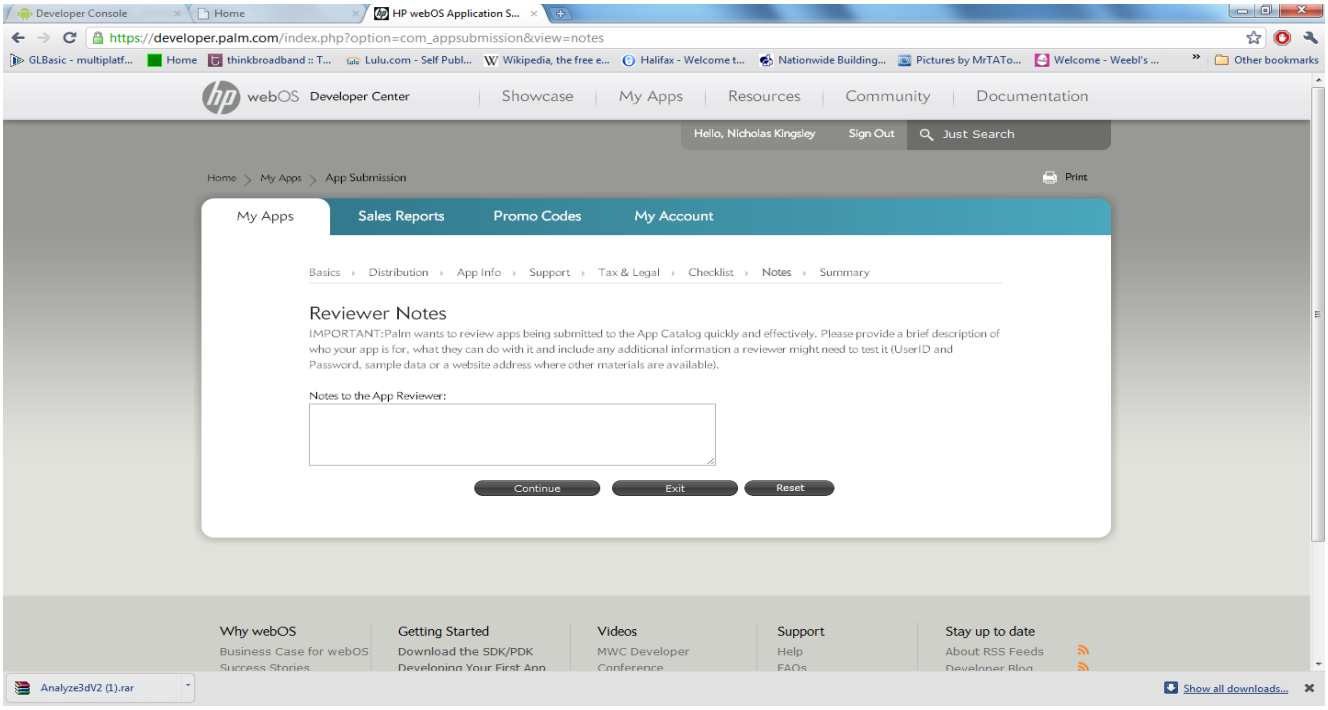
This section you need to mention whether your application uses location tracking or data encryption (due to US rules). You also need to enter a copyright statement and a URL for your license agreement (if any)

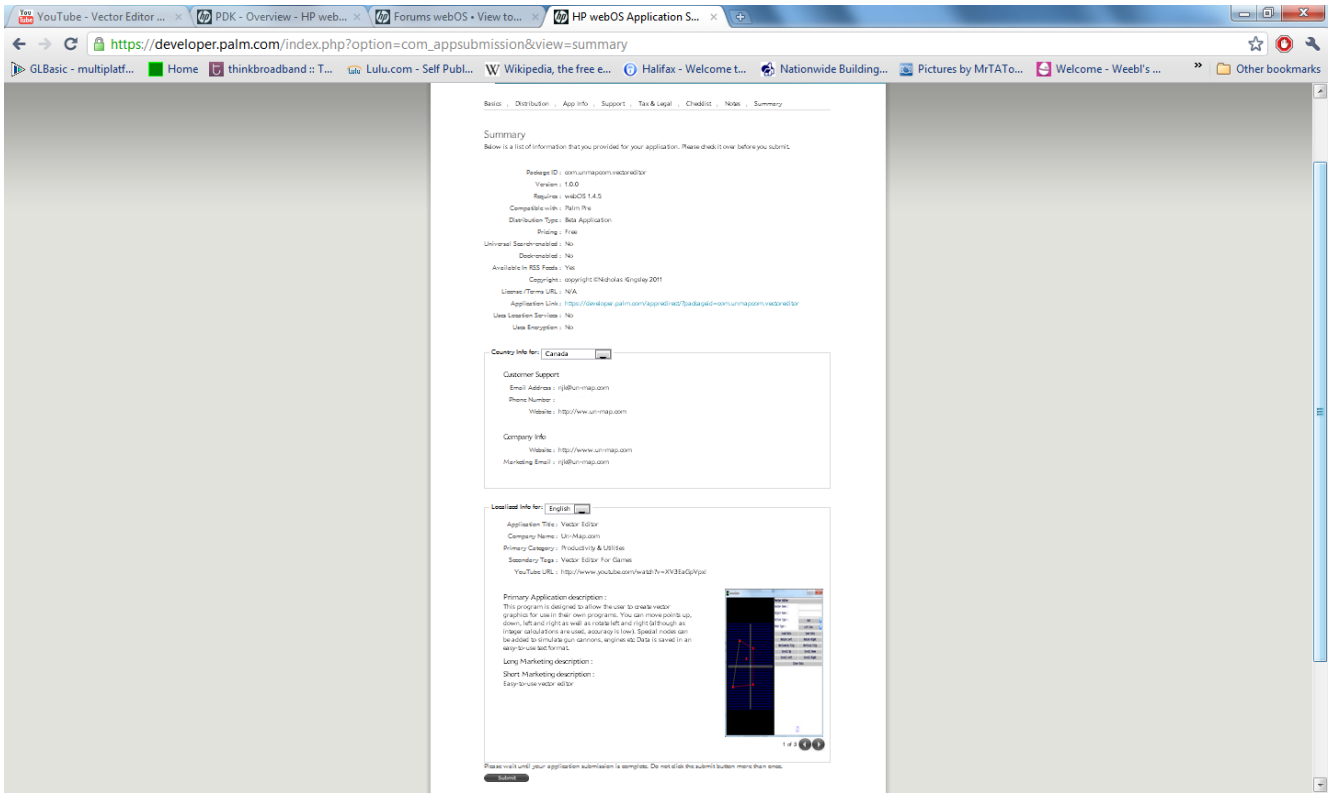

HP webOS Application S...
Home
https://developer.palm.com/index.php?option=com_appsubmission&view=certification
webOS Developer Center
Showcase
My Apps
Resources
Community
Documentation
Hello, Nicholas Kingsley
Sign Out
Just Search
Home
My Apps
App Submission
Print
My Apps
Sales Reports
Promo Codes
My Account
Basics
Distribution
App Info
Support
Tax & Legal
Checklist
Notes
Summary
Checklist
You must agree to all of the items below. These terms are legally binding. Palm reserves the right to exclude an application if it deems it is otherwise inappropriate for Palm users.
Application Publication Policy
In order for an application to be published in the App Catalog, the application must meet all of the criteria set forth in this Application Publication Policy. The Policy sets forth guidelines on the Content of the application, Rules of Good Behavior, Technical Criteria, User Interface Criteria and certain Other Legal Requirements
In order to publish your Application in the App Catalog, you must certify that the Application meets all of the criteria set forth in this Policy. By clicking on the "Certify" button below, you are making legally binding certifications to HP. HP reserves the right to exclude an Application from being published in the App Catalog for any reason in its sole discretion.
Content Criteria
Section 1: Content Appropriateness
Select All
Clear All
PDF
Continue
Exit
Reset
Why webOS
Getting Started
Videos
Support
Stay up to date

This is a general list of what is and isn't acceptable for submission, which should be read. At the bottom of this screen is a tick-boxed which would need to be ticked to confirm that you understand the submission criteria

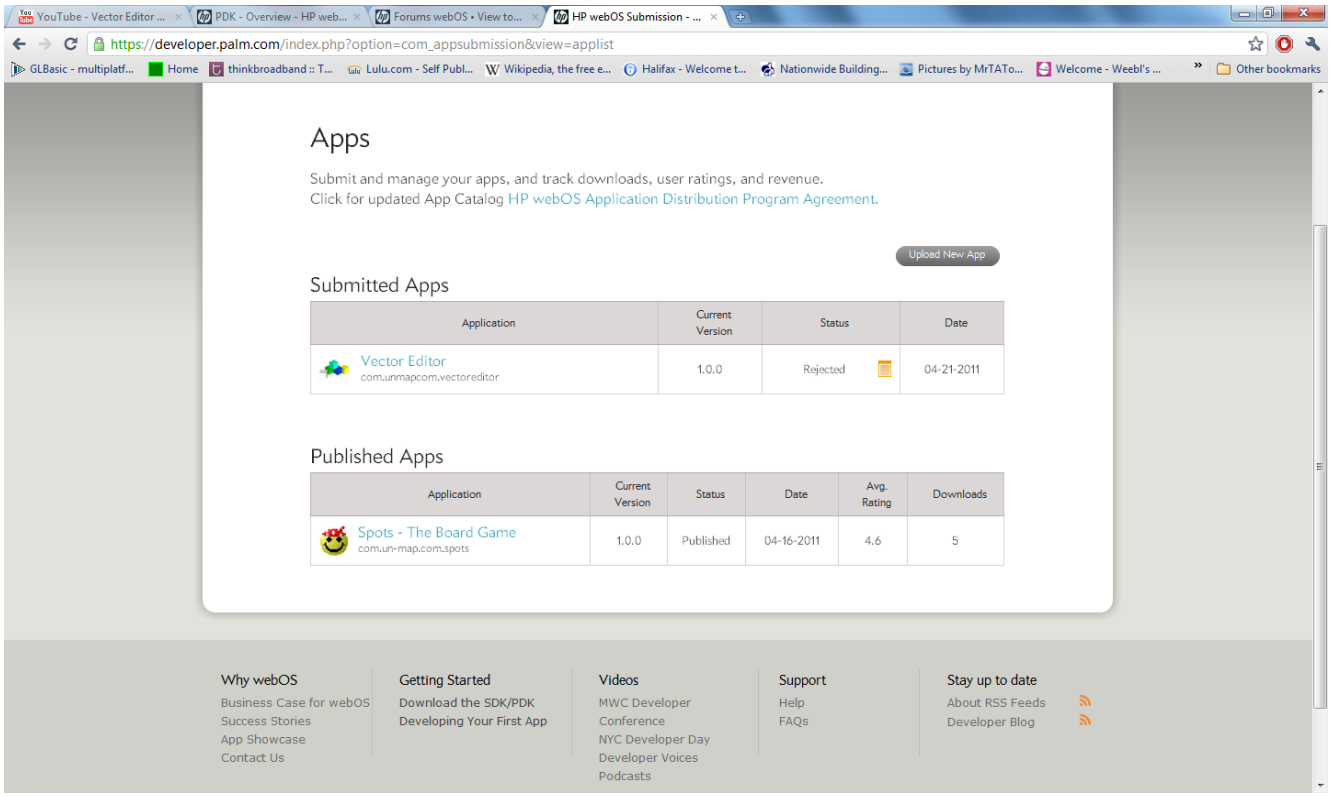


For beta applications, this section isn't filled out, although it is for store and web submission, and looks like :

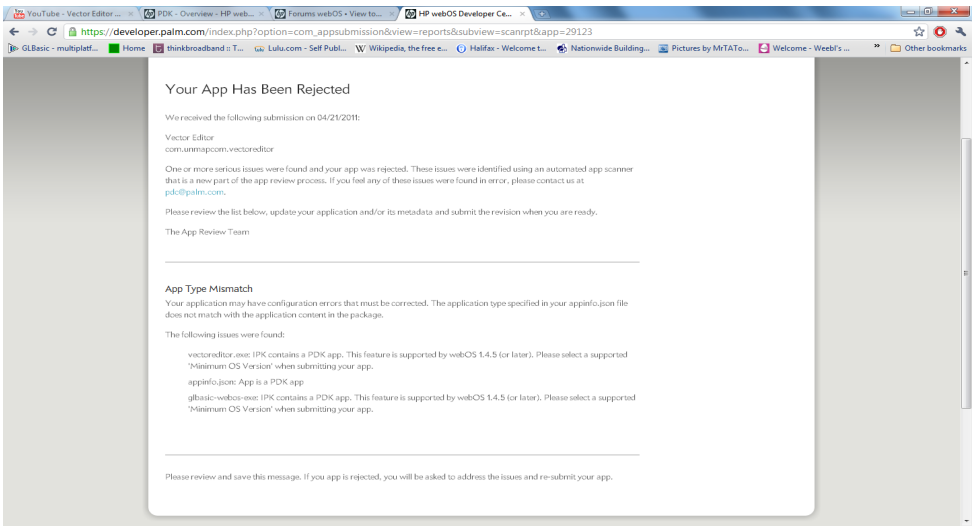

Developer Console
Home
HP webOS Application S...
https://developer.palm.com/index.php?option=com_appsubmission&view=notes
webOS Developer Center
Showcase
My Apps
Resources
Community
Documentation
Hello, Nicholas Kingsley
Sign Out
Just Search
Home > My Apps > App Submission
Print
My Apps
Sales Reports
Promo Codes
My Account
Basics › Distribution › App Info › Support › Tax & Legal › Checklist › Notes › Summary
Reviewer Notes
IMPORTANT:Palm wants to review apps being submitted to the App Catalog quickly and effectively. Please provide a brief description of who your app is for, what they can do with it and include any additional information a reviewer might need to test it (UserID and Password, sample data or a website address where other materials are available).
Notes to the App Reviewer:
Continue
Exit
Reset
Why webOS
Business Case for webOS
Getting Started
Download the SDK/PDK
Videos
MWC Developer
Support
Help
Stay up to date
About RSS Feeds
Analyze3dV2 (1).rar
Show all downloads...


YouTube - Vector Editor ...
PDK - Overview - HP web...
Forums webOS • View to...
HP webOS Application S...
https://developer.palm.com/index.php?option=com_appsubmission&view=summary
GLBasic - multiplatf...
Home
thinkbroadband :: T...
Lulu.com - Self Publ...
Wikipedia, the free e...
Halifax - Welcome t...
Nationwide Building...
Pictures by MrTATo...
Welcome - Weebl's ...
Other bookmarks
Basics , Distribution , App Info , Support , Tax & Legal , Checklist , Notes , Summary
Summary
Below is a list of information that you provided for your application. Please check it over before you submit.
Package ID : com.unmapcom.vectoreditor
Version : 1.0.0
Requires : webOS 1.4.5
Compatible with : Palm Pre
Distribution Type : Beta Application
Pricing : Free
Universal Search-enabled : No
Bookmarkable : No
Available in RSS Feeds : Yes
Copyright : copyright ©Nicholas Kingsley 2011
License /Terms URL : N/A
Application Link : https://developer.palm.com/appredirect/?packageid=com.unmapcom.vectoreditor
Uses Location Services : No
Uses Encryption : No
Country Info for: Canada
Customer Support
Email Address : nj[@]un-map.com
Phone Number :
Website : http://www.un-map.com
Company Info
Website : http://www.un-map.com
Marketing Email : nj[@]un-map.com
Localized Info for: English
Application Title : Vector Editor
Company Name : Un-Map.com
Primary Category : Productivity & Utilities
Secondary Tags : Vector Editor For Games
YouTube URL : http://www.youtube.com/watch?v=XV3EeQpVpoI
Primary Application description :
This program is designed to allow the user to create vector graphics for use in their own programs. You can move points up, down, left and right as well as rotate left and right (although as trilogy calculations are used, accuracy is low). Special nodes can be added to simulate gun cannons, engines etc. Data is saved in an easy-to-use text format.
Long Marketing description :
Short Marketing description :
Easy-to-use vector editor
1 of 3
Please wait until your application submission is complete. Do not click the submit button more than once.
Submit

This is an overview of everything that has been entered



Until recently, all PDK applications would be automatically rejected (due to a quirk in the automated submission process), forcing you to email Palm (using the contact address given in the rejection email) to convert the program to an "accepted" status.


https://developer.palm.com/index.php?option=com_appsubmission&view=reports&subview=scanrpt&app=29123
Your App Has Been Rejected
We received the following submission on 04/21/2011:
Vector Editor
com.unmapcom.vectoreditor
One or more serious issues were found and your app was rejected. These issues were identified using an automated app scanner that is a new part of the app review process. If you feel any of these issues were found in error, please contact us at pdc@palm.com.
Please review the list below, update your application and/or its metadata and submit the revision when you are ready.
The App Review Team
App Type Mismatch
Your application may have configuration errors that must be corrected. The application type specified in your appinfo.json file does not match with the application content in the package.
The following issues were found:
vectoreditor.exe: IPK contains a PDK app. This feature is supported by webOS 1.4.5 (or later). Please select a supported 'Minimum OS Version' when submitting your app.
appinfo.json: App is a PDK app
glbasic-webos-exe: IPK contains a PDK app. This feature is supported by webOS 1.4.5 (or later). Please select a supported 'Minimum OS Version' when submitting your app.
Please review and save this message. If you app is rejected, you will be asked to address the issues and re-submit your app.

This is the rejection message you could receive. However, PDK applications should now be accepted unless there is an actual problem with the submitted package.

It generally takes around 5 – 7 days to evaluate a package for submission to the Palm Catalog, and a few days less for beta submissions.

A submission failure will force you to wait the full time again – failed applications aren't fast-tracked through the submission process.

One note of warning : If an application is submitted for the beta program, and then at a later date the same application is submitted to the standard catalogue, then you will not be able to charge for it (as beta applications are always free and can never be changed to paid ones).

Compiling for Android

Compiling for Android is not as straight forward as the other platforms, as it requires the downloading (for development purposes only) of two other SDK's – the Java SDK and Android SDK

The Java SDK can be found at http://www.oracle.com/technetwork/java/javase/downloads/index.html, whilst the Android SDK is at http://developer.android.com/sdk/index.html

The Java SDK (only use the 32-bit version) can be installed anywhere (and can have spaces in the path name), and needs to be done before the Android SDK is installed.

Once done, you will need to add **JAVA_HOME** environment variable to point to the directory containing the version. So, for example, if the version 6.25 of the JDK was installed in *C:\Program Files (x86)\Java*, **JAVA_HOME** would be set to *C:\Program Files (x86)\Java\jdk1.6.0_25*

This path should also be appended to the **PATH** variable too

Once done, the Android SDK can now be installed. This should do in a directory path that has no spaces (so the usual *Program Files* directory is out!). Usually on the first run, it will say that the Java SDK couldn't be found, but if you click on **BACK** and then **NEXT** again, it will be magically okay.

Before installing, make sure everything is selected.

Once completed, you have one more environment variable to setup. This is called **ANDROID_SWT** and should be set to the 32-bit version of **swt.jar** in the *android-sdk\lib* directory. So, for example, if the Android SDK was installed in *C:\Android*, **ANDROID_SWT** would be set to *C:\Android\android-sdk\tools\lib\x86*

As a GLBasic program can run on an Android emulator, you may want to test your programs on that first. It should be noted that a lot of extensions are not supported, so the display may not be the same as an actual device.

To install an application on the emulator, you use (assuming **%PATH%** points to the Android program directory) :

```
adb install your_application.apk
```

With your *your_application.apk* pointing to the APK file that you want installed.

You can also grab emulator screenshots if you run *ddms* before you start the emulator.

Android Marketplace

Submitting an application to the Android Marketplace is just as simple as a webOS submission, although you do have to pay (at the time of writing) $25 (around 16.90 Euros or £14.98).

In addition, before you can submit a program, you have to sign it with a developer key. To do so, follow these steps :

- Build your application for Android
- Make sure your JAVA paths are correct and setup in **%PATH%**
- Create a folder called *keytools* and make a folder in that called *keys*. The best place to do so would be the root of your GLBasic project folder
- Copy the **glbasic-debug.apk** file in your projects *android/bin* directory, into the keytools directory.

As the APK file has been signed with a default key, this needs to be removed and replaced with one acceptable for the Marketplace. This means using an application like WinRAR or 7ZIP to remove the contents of *META-INF* directory. Once done, the code-signing process can continue :

- Open a command prompt (CMD.EXE)
- Move to the keytools folder (for example using C:\Users\Nicholas\Documents\GLBasic\Keytools)

Type in : `keytool -genkey -alias mykey.keystore -keyalg RSA -validity 20000 -keystore keys/mykey.keystore`

You will then be asked the following questions (all of which must be answered) :

- Password
- Full name
- Name of your organisational unit
- Name of your organisation
- Name of City/Locality
- State/Province
- Two letter country code

After which, you can press RETURN to use the standard key. Once you are back at the command prompt, type in the following : `jarsigner -verbose -keystore keys/mykey.keystore -signedjar glbasic-signed.apk glbasic-debug.apk mykey.keystore`

Your application will now be signed. At this point, it would also be a good idea to zipalign the APK file. This can be done by typing in :

```
zipalign -v 4 glbasic-debug.apk glbasic-signed.apk
```

which will output something like :

```
Verifying alignment of glbasic-signed.apk (4)...
      60 assets/Media/smalfont.png (OK)
    3944 res/drawable/icon.png (OK)
    8560 res/layout/main.xml (OK - compressed)
    8910 AndroidManifest.xml (OK - compressed)
    9652 resources.arsc (OK)
   10537 classes.dex (OK - compressed)
   20262 lib/armeabi/gdbserver (OK - compressed)
   94389 lib/armeabi/libmain.so (OK - compressed)
  596294 lib/armeabi/libmikmod.so (OK - compressed)
  699032 lib/armeabi/libSDL.so (OK - compressed)
  906556 lib/armeabi/libSDL_image.so (OK - compressed)
 1008474 lib/armeabi/libSDL_mixer.so (OK - compressed)
 1108476 lib/armeabi/libSDL_ttf.so (OK - compressed)
Verification succesful
```

A batch script to do all the extracting and re-signing is as follows (you will need to install 7zip for it to work) :

```
copy %1\glbasic-debug.apk glbasic-debug.zip
"C:\Program Files (x86)\7-Zip\7z.exe" d glbasic-debug.zip META-INF\manifest.mf META-INF\CERT.SF META-
INF\CERT.RSA
copy glbasic-debug.zip glbasic-debug.apk
keytool -genkey -alias mykey.keystore -keyalg RSA -validity 20000 -keystore keys/mykey.keystore
jarsigner -verbose -keystore keys/mykey.keystore -signedjar glbasic-signed.apk glbasic-debug.apk
mykey.keystore
zipalign -v 4 glbasic-signed.apk glbasic-release.apk
```

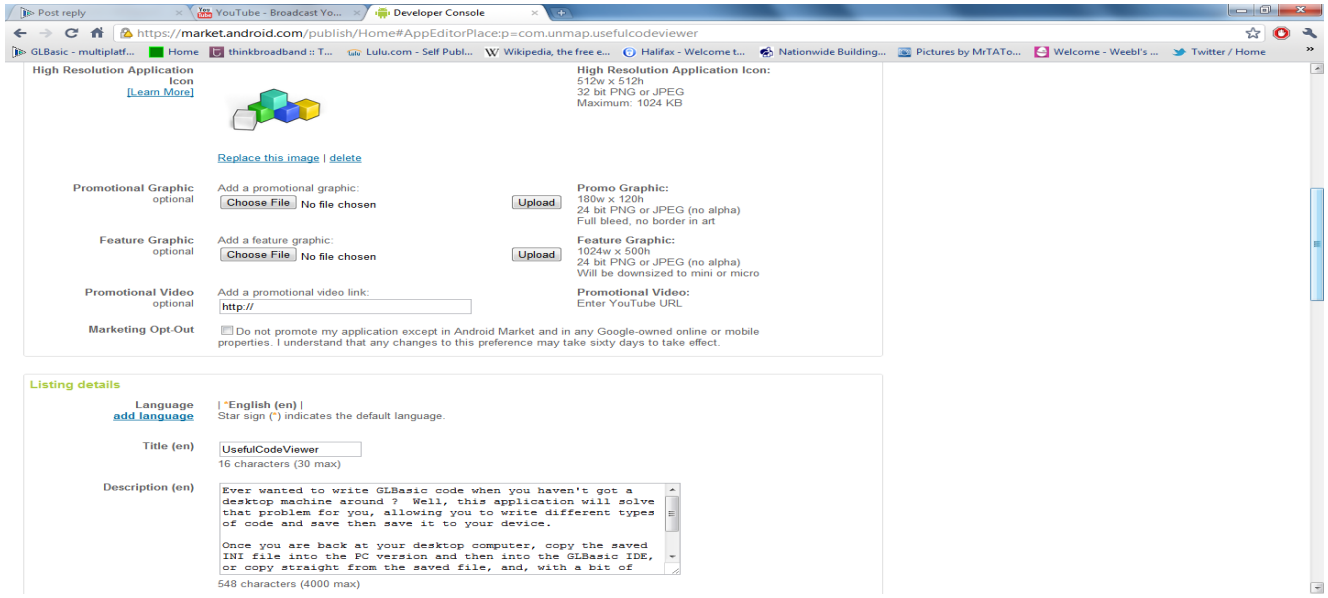
When you have paid your Marketplace fees, you can submit your application to the store. Acceptance is almost instant, so there is little in the way of application review before it is accepted.

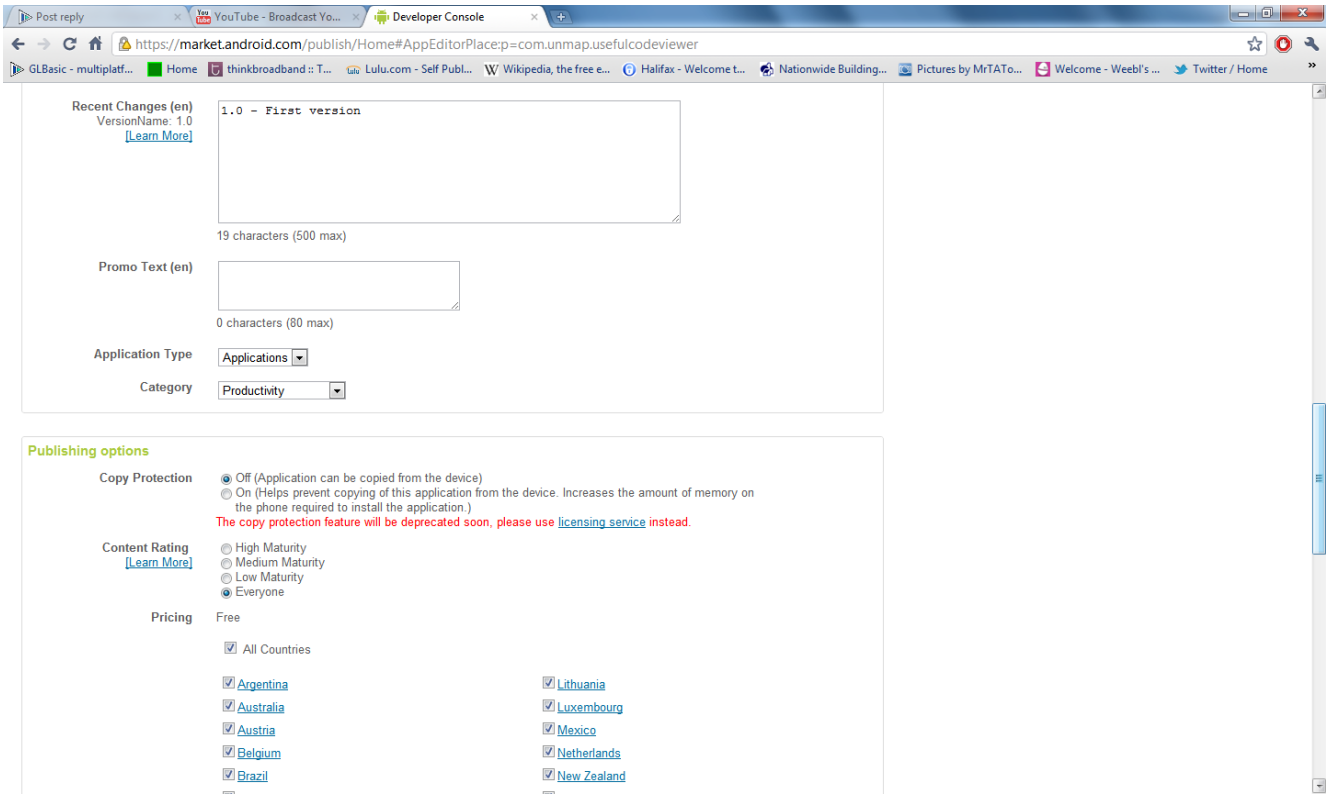
In addition to the standard Android Market, it might also “pay” to submit it to the 1mobile.com site as well. To do so, create an author account at http://author.1mobile.com/

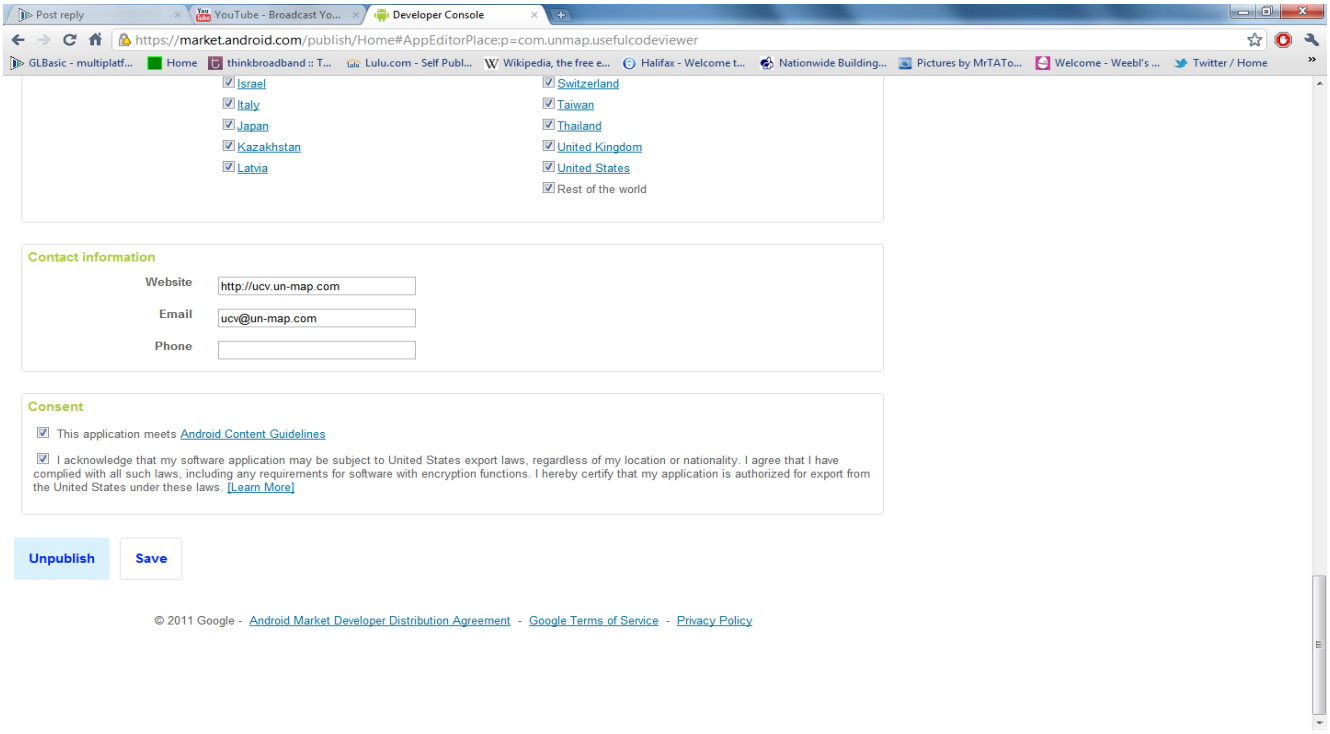
If you find that the Clean Project option isn't working (Windows stating that a file is locked), you will need to open a command prompt and type :

```
adb kill-server
```

The Marketplace looks like :


Post reply
YouTube - Broadcast Yo...
Developer Console
https://market.android.com/publish/Home#AppEditorPlace:p=com.unmap.usefulcodeviewer
High Resolution Application Icon
[Learn More]
High Resolution Application Icon:
512w x 512h
32 bit PNG or JPEG
Maximum: 1024 KB
Replace this image | delete
Promotional Graphic
optional
Add a promotional graphic:
Choose File No file chosen
Upload
Promo Graphic:
180w x 120h
24 bit PNG or JPEG (no alpha)
Full bleed, no border in art
Feature Graphic
optional
Add a feature graphic:
Choose File No file chosen
Upload
Feature Graphic:
1024w x 500h
24 bit PNG or JPEG (no alpha)
Will be downsized to mini or micro
Promotional Video
optional
Add a promotional video link:
http://
Promotional Video:
Enter YouTube URL
Marketing Opt-Out
Do not promote my application except in Android Market and in any Google-owned online or mobile properties. I understand that any changes to this preference may take sixty days to take effect.
Listing details
Language
add language
| *English (en) |
Star sign (*) indicates the default language.
Title (en)
UsefulCodeViewer
16 characters (30 max)
Description (en)
Ever wanted to write GLBasic code when you haven't got a desktop machine around ? Well, this application will solve that problem for you, allowing you to write different types of code and save then save it to your device.
Once you are back at your desktop computer, copy the saved INI file into the PC version and then into the GLBasic IDE, or copy straight from the saved file, and, with a bit of
548 characters (4000 max)




Post reply
YouTube - Broadcast Yo...
Developer Console
https://market.android.com/publish/Home#AppEditorPlace:p=com.unmap.usefulcodeviewer
GLBasic - multiplatf...
Home
thinkbroadband :: T...
Lulu.com - Self Publ...
Wikipedia, the free e...
Halifax - Welcome t...
Nationwide Building...
Pictures by MrTATo...
Welcome - Weebl's ...
Twitter / Home
Israel
Italy
Japan
Kazakhstan
Latvia
Switzerland
Taiwan
Thailand
United Kingdom
United States
Rest of the world
Contact information
Website
http://ucv.un-map.com
Email
ucv@un-map.com
Phone
Consent
This application meets Android Content Guidelines
I acknowledge that my software application may be subject to United States export laws, regardless of my location or nationality. I agree that I have complied with all such laws, including any requirements for software with encryption functions. I hereby certify that my application is authorized for export from the United States under these laws. [Learn More]
Unpublish
Save
© 2011 Google - Android Market Developer Distribution Agreement - Google Terms of Service - Privacy Policy

Movement (and animations)

In order to make sure that sprite moment is more or less the same no matter what processor type or speed, there are several things you can try :

- Fix frame rate to some limit
- Add in a SLEEP command that is executed every loop
- Link movement speed and when needed, animation to the speed at which a sprite moves (or animates) an an elapsed time

The last option is the best way and allows consistent movement (and animations) across all supported platforms.

There are many different ways of doing this, the simplest being (being written by Hemlos) :

```
GETSCREENSIZE ScreenX , ScreenY
TimeElapsed=GETTIMER()/1000
IF TimeElapsed>1.0 THEN TimeElapsed=1.0   // prevent out of control movement
IF TimeElapsed<0.0001 THEN TimeElapsed=0.0001   // prevent out of control movement
MovementSegment =  TimeElapsed * SQR( ScreenY * ScreenY * 2 ) //hypot of a right triangle
```

Fading Up and Down

Sometimes you may want to fade text and graphics up and/or down. For full screen fading down, you could use BLENDSCREEN, but for fading up or for specific graphics, you have to write your own system.

Things are slightly complicated by the fact that a fully solid and invisible graphic isn't in a simple range, instead an almost invisible object is around -0.01, whilst a solid shape is 0.0, 1.0 and -1.0

The easiest way to fade down is to use something like :

```
LOCAL loop

FOR loop=-1.0 TO -0.01 STEP 0.01
        ALPHAMODE loop
        PRINT "Hello world",0,0
        SHOWSCREEN
NEXT

WHILE TRUE
        ALPHAMODE loop
        PRINT "Hello world",0,0
        SHOWSCREEN
WEND
```

Which slowly fades text from white to black (and thus invisible)

To fade text up to white, its just a matter of reversing this process :

```
LOCAL loop

FOR loop=-0.01 TO -1.0 STEP -0.01
        ALPHAMODE loop
        PRINT "Hello world",0,0
        SHOWSCREEN
NEXT

WHILE TRUE
        ALPHAMODE loop
        PRINT "Hello world",0,0
        SHOWSCREEN
WEND
```

Portrait and Landscape

iPhone/iPod/iPad

On an iPhone/iPod/iPad, there are currently four screen resolutions :

- 320 x 480
- 480 x 320
- 768 x 1024
- 1024 x 768

320 x 480 and 768 x 1024 are regarded are Portrait modes (as the screen height is greater than the width), whilst 480 x 320 and 1024 x 768 are regarded as Landscape modes

There are a few commands in GLBasic that wont work in Landscape mode and a few that might be slower in Landscape mode than Portrait.

Palm Pre

The Palm Pre currently has two screen resolutions :

- 320 x 480
- 480 x 32

Android

Android machines come in a variety of resolutions :

- 320 x 480
- 480 x 320
- 480 x 800
- 800 x 480
- 600 x 800
- 800 x 600

Icons and Graphics

After the first compilation, in the project directory you will find a file "icon.png". This file is a 24 bit png image. The colour `RGB(255,0,128)` will be considered transparent (alpha information will be ignored!). You can use any size for this image but a square shape is recommended (i.e. equal height and width).

If you now create the program for a certain platform, the icon will be converted to the required format and copied to the right place. Note : that this is only performed if the target (icon) file does not already exist in the .app directory. Thus, you can manually edit the output (icon) file and GLBasic will not overwrite it any more.

For Windows platforms, there is a special feature. The "icon.png" will be converted to an "icon.ico" file in the project directory. This icon will be appended after each linking.

The required sizes of icons and other graphics is as follows :

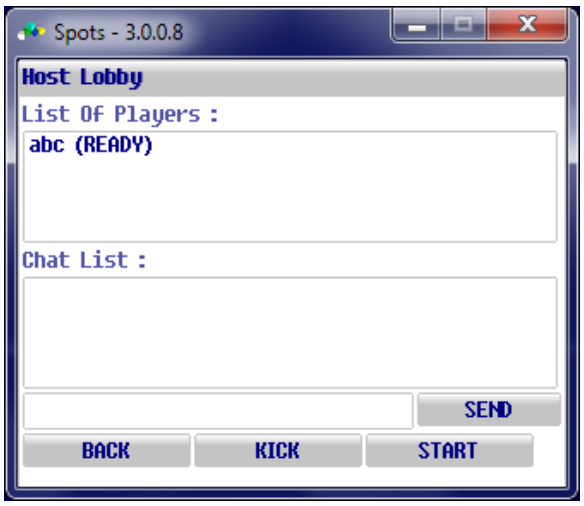
Platform	Icons and Graphics needed
Windows	Standard ICO file
Linux	Standard application icon
Mac	Standard Icns file
iPhone	57 x 57, 29 x 29, 512 x 512
webOS	48 x 48, 64 x 64, 640 x 480
Android	72 x 72, 512 x 512
Windows CE	None
GP2X-WIZ/Caanoo	Standard program icon
GP2X	Standard program icon
Pandroa	Standard program icon

Networking Lobby

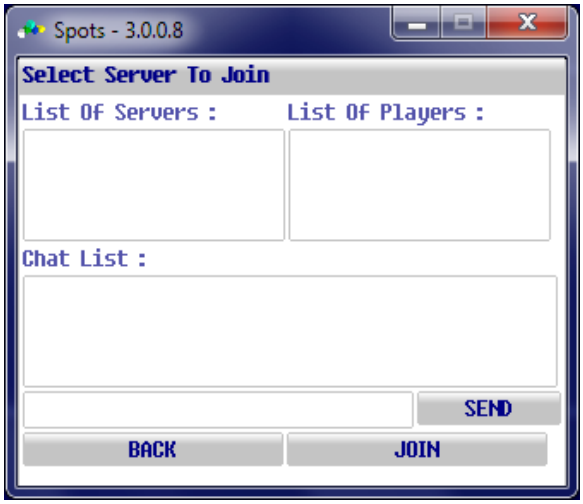
A rarely discussed topic, even in networking books, is how to create a networking lobby. This section will detail a basic local area network lobby, which could be modified to your specifications – the code can't be used as it is, due to the use of the TSetup type. However, if that is removed (and all code that references `setup.returnScreenWidth()` and `setup.returnScreenHeight` is replaced), then the following routine should work very well!

Please contact the author is the code cannot be found in the GLBasic forums

This is what the host of a lobby looks like :



And this is the client :



A networking lobby allows clients to join a network game without the need to enter an IP address or domain name – instead it's just a matter of selecting a lobby and joining it, making the whole process easier than if a manual IP address had to be entered.

Of course, all this convenience does require a lot of coding effort, but does make your software more professional.

So, how does it all work ?

Broadcasting

The most important part is the broadcasting system. This is needed to discover appropriate servers, and can only be done using UDP sockets.

As you will see in the code below, the client machine sends out a broadcast message to which the servers respond. Whilst the other way around could be done (servers send a broadcast message to which clients respond), I feel the former way is a bit better as it reduces the amount of server code.

Broadcasting is performed by sending the IP address of “255.255.255.255”. It should be noted that this IP address only works on the network the computer is attached to. It should be noted that all computers on this network will receive the message, including the sender!

A server will respond with appropriate information.

Reliable V Unreliable messages

UDP messages suffer from the fact that messages may not arrive, arrive just the once or perhaps arrive many times, and thus TCP's reliable messaging system (where data is retransmitted automatically if there is a problem) needs to be implemented.

To do this a reliable messaging routine is set-up where a message is stored, and, if a suitable acknowledgement isn't received within a certain time, the original message is re-transmitted. If, after a certain number of times there is no reply to the message, then the machine to which the message should be sent is regarded as being disconnected and appropriate action is taken.

Not all messages need to be reliable (broadcasting and chat messages especially), and so the network data isn't always stored.

Timing

A 5000 ms re-try time seems to be suitable. It is also the same amount of time used to send broadcast messages, although you can change these values if needed.

Limitations

Windows Limitations

- OGG files requires the relevant codec to be installed
- Error log is created as %TEMP%/ErrorLog.txt
- Minimum requirements is Windows 2000 (with Service Pack 4) upwards. Will run on NT4 too (up to V8.078). V9 is no longer compatible with Windows 2000 or NT
- Microsoft Foundation Class 2005 will be installed during program installation, and may be needed by compiled programs if run on a different machine
- When changing screen resolutions, all sounds and graphics need to be reloaded

Mac Limitations

- `FILEREQUEST$` is replaced by `INPUT$`
- Works on PowerPC or Intel machines (Universal executables are created)
- Full screen mode is currently unavailable
- MP3 files should work (in addition to OGG)
- Cannot change to full screen mode after using window mode
- When changing screen resolutions, all sounds and graphics need to be reloaded

iPod/iPhone/iPad Limitations

- `FILEREQUEST$` is replaced by `INPUT$`
- 32-bit floating-point numbers are used
- Display could be 16-bit
- There is no keyboard, so `KEY()` returns 0
- Multi-touch is supported. Each press on the screen is defined as a mouse device
- If the HOME button is pressed, the operating system will exit your program
- The following commands are unsupported : `BLENDSCREEN` and `USEASBMP`
- No spaces should be in the project name - if there are, XCode wont compile the project.
- The acceleration sensor of the iPhone can be polled with `GETJOYX(0)`, `GETJOYY(0)` and `GETJOYZ(0)` accordingly. The return values are the accelerations for each axis in "G" (9.81 m/s^2).
- Version 8 supports OS 4 multi-tasking

Apple doesn't allow you to install parts of the XCode SDK on a non-Apple branded machine.

To avoid problems, GLBasic creates an XCode project that must be compiled on a Mac
The code-signed program file must be copied to the .app directory and called "iPhone". An iMac MUST be used to compile the project. The XCode development system MUST also be installed
If you want to develop for the iPod/iPhone/iPad, you need to sign up to Apple's Developer Program.

Windows CE Limitations

- FILEREQUEST$ is replaced by INPUT$
- 32-bit floating-point numbers are used
- Display could be 16-bit
- The mask in PNG files is ignored. Make the bitmaps 24 bit and set a transparency colour
- Screen resolutions can be highly variable. Quality of 3D output is variable
- With V8, Windows CE netbooks are supported
- Need to set screen resolution to the target machine, otherwise graphics may be corrupted

GP32/2X/WIZ/Caanoo Limitations

- FILEREQUEST$ is replaced by INPUT$
- 32-bit floating-point numbers are used
- Display could be 16-bit
- Screen resolution is 320 x 240
- Memory is limited to around 48Mb, except for the Caanoo which has slightly more
- MIDI playback may not work

XBox Linux Limitations – Is available as a hidden option only

- FILEREQUEST$ is replaced by INPUT$
- Linux for the XBox needs to be installed
- Wont work in standard Linux installations like Ubuntu (segment faults)

Pandora Limitations

- Before V9.040, programs needed to be compiled on the machine itself. However, from 9.040, GLBasic will create the executables itself.

Android Limitations

- Requires at least OS 2.2
- Generic and earlier devices may not support all the OpenGL extensions required by GLBasic, possibly resulting in corrupt graphics and white lines. This is also true for the emulator. DRAWLINE and DRAWRECT are affected by this
- It could also affect music playing too
- Android programs should work on replacement operating systems. Only Uberoid 2.2 has been tested, and programs work fine on it.
- TRY/CATCH/THROW will cause the program to exit (when THROW is called)

Palm Pre/Tablet/Pixi Limitations

- Palm Pixi screen resolution is 400 x 400 (somewhat smaller than the Pre)
- The Palm Pre forwards all input actions to the GLBasic program Additionally the back gesture is submitted as `KEY(1)` = **ESC**. Here, you must set `ALLOWESCAPE FALSE`, otherwise the program would just end.
- Vibration is only accessible in WebOS 2.0.x and thus not supported, yet.
- The axes are returned with the `GETJOY` commands in the same way as on the iPhone! (Internally the Palm Pre has the Y axis mirrored).
- The Palm Pre has a screen resolution of 320 x 480 and 480 x 320.
- Minimum Smartphone version is 1.4.5, whilst the Tablet uses webOS 3
- GLBasic programs currently won't work on the emulator (nothing happens)
- The Pre supports up to 5 touches. The Tablet is apparently supposed to support up to 16
- Bluetooth keyboard currently uses scancode values that GLBasic doesn't detect

Linux Limitations

- `FILEREQUEST$` is replaced by `INPUT$`
- Cannot change to full screen mode after using window mode
- When changing screen resolutions, all sounds and graphics need to be reloaded

Installation instructions for GLBasic before V10

The following libraries need to be installed before running an executable.

libSDL_mixer-1.2
libstdc++.so.5
libmikmod.so.2
libvorbisfile.so.3
libsmpeg-0.4.so.0
libvorbis.so.0

With Ubuntu 9.10 (or slightly earlier), this can be done with :

```
sudo apt-get install libsdl-mixer1.2-dev libstdc++5
```

However with later versions of Ubuntu, these libraries are unable using the above command, and need to fetched using an alternative way. For example :

Install getlibs from http://ubuntuforums.org/showthread.php?t=474790

and in Terminal type :

```
getlibs -l libmikmod.so.2 libvorbisfile.so.3 libsmpeg-0.4.so.0 libvorbis.so.0
```

Enter your admin password and press Y to install all dependencies.

```
cd ~/Desktopwget http://frozenfox.freehostia.com/cappy/getlibs-all.deb
wget http://nl.archive.ubuntu.com/ubuntu/pool/universe/g/gcc-3.3/libstdc++5_3.3.6-17ubuntu1_i386.deb
sudo dpkg -i getlibs-all.debgetlibs -i libstdc++5_3.3.6-17ubuntu1_i386.deb
```

This works fine with Ubuntu 10 and later

These packages can be downloaded from the GLBasic website at :
http://www.glbasic.com/files/linux_deb.rar

The HappyPenguin website (http://happypenguin.org/) is a good place to submit your Linux programs.

Installation instructions for GLBasic after V10

From V10 onwards, all that should be needed to run a Linux application is to install SDL Mixer (using t he `sudo apt-get` command described on the previous page.

Note about Pathing

You should be careful when using a "\" character as a directory or drive separator. This is because a single \ denotes the next character will be an escape code (to allow things like new lines). For this reason, either use a forward slash character (/) or two backslashes (\\) as a drive and directory separator.

Programming Guide Lines

Whilst programming styles are individualistic, there are some rules that should be obeyed to make sure that your code can be understood easily, both in the present and in the future :

No to GOTO

There just isn't a need nowadays to use `GOTO` - its ability to branch can be dealt with in a more understandable way using other commands like `REPEAT`/`UNTIL`/`WHILE`/`END`/`FUNCTION`

Comments

Each separate module (whether its a function or subroutine) should be documented in a concise and accurate manner. However, there is no need to write a complete biography on what the routine does - keep it short and too the point.

Each module should detail who wrote it and when, with subsequent modifications being noted and detailed - but that is only really needed if there is more than one programmer.

Movement

All movement should be independent of a set speed or processor speed - whilst it may be fine on your machine, it could run slower than a tortoise in treacle on someone else's machine or faster than a rabbit with a jet-pack.

Testing

It always helps if you can test your program on machines completely different from your development system. If you can't do this, then the next best thing is to use a virtual machine with different operating systems and memory configurations.

Your code should always gracefully exit if there is a problem

Don’t re-invent the wheel

If something that you need to do is already present, then use it - don't bother re-inventing the wheel as it just takes time and effort that can be placed elsewhere more productively
Re-use code

Try to write code so that it can be re-used for future projects with no need for modifications - just additions

Guest Account

Make sure that your program works properly when it has limited access to the hard drive and operating system.

Don’t change users settings

Don’t go around changing any of the user's settings or backgrounds. And of course, don’t delete any files!

Style Guidelines

General ideas to keep your users happy...

Let Users Know What Is Happening

People don't like sitting at blank screens for any length of time. How do they know if the program is actually still working and hasn't crashed ?
Make sure people are fully informed during long processes - use progress bars and so on

Don't Alarm the User

If a problem occurs, don't just end a program - display a user-friendly error message that doesn't alarm them.

If possible, try and save information to allow resumption at near enough the same position, if possible.

Don't Make Assumptions

Don't assume anything about a potential user's machine, beyond the fact that your program is being used on the correct operating system...

Be user-friendly

Where appropriate try to take into account as many control/input systems as usual - perhaps the user doesn't have a mouse and so will have to use the keyboard or vice-versa...

Your program may also be used for the visually impaired - you might need to produce high-contrast fonts and graphics

Internationalise

If possible, produce text for various countries, in-case of users who don't speak your language
Leave the OS in the state before your game started. The DDgui Programming Guide contains a useful section on Internationalisation

Free all used memory

Whilst most operating systems should do this after your application finishes, it can't always be relied upon, and you should release all used memory (unload sprites, sound effects etc.)

Allow people to skip bits if possible

People don't want to always sit through a 10 minute intro. sequence - allow them to easily skip it.

Installation Problems

Occasionally, you might find (especially on Vista and Windows 7 machines), that, during the installation of Visual Studio 2005, something like the following error message is displayed :

**Activation context generation failed for "C:\Program Files (x86)\GLBasic\EditorE.exe".
Dependent Assembly Microsoft.VC80.MFC,processorArchitecture="x86",
publicKeyToken="1fc8b3b9a1e18e3b", type="win32",version="8.0.50727.4053"
could not be found.**

This usually occurs due to either a conflict with a previous version of Visual Studio 2005 or because you are installing another program at the same time.

The installation of Visual Studio 2005 has to be aborted otherwise the installation of GLBasic will never finished. At this point, the editor can never be run without the above error message appearing.

To solve this problem, you will need (in the GLBasic\Tools directory) to run the vcresdist_x86 after GLBasic has finished installing, and after any other programs you are installing at the same time have finished.

The Visual Studio 2005 re-distributable should then install correctly, allowing full use of GLBasic. If however it doesn't you may need to location a newer version on the Microsoft website.

From V9.040 onwards, the Visual Studio 2010 re-distributable package will need to be installed instead of 2005.

Error Messages

GLBasic can output two sets of error messages. The first and more common sort are those in your native language (and thus should be easy to understand). The second lot are those returned by the c++ compiler and are generally harder to solve.

The standard error messages are :

Error Message :	Meaning :
FOR without NEXT	A FOR statement doesn't have a corresponding NEXT statement in the same code block
NEXT without FOR	A NEXT statement doesn't have a corresponding FOR statement in the same code block
WHILE without WEND	A WHILE statement doesn't have a corresponding WEND statement in the same code block
WEND without WHILE	A WEND statement doesn't have a corresponding WHILE statement in the same code block
UNTIL without REPEAT	An UNTIL statement doesn't have a corresponding REPEAT statement in the same code block or a REPEAT statement doesn't have a corresponding UNTIL statement in the same code block
call to undefined function	The program is trying to call a function that doesn't exist
wrong number or (*sic*) arguments	The incorrect number of arguments (too many or too few) are being passed to a function

wrong argument type	The wrong type of value is being passed to a function. For example, the following code :
	
	`rep("Y",1)`
	
	`FUNCTION rep%:a AS byte,b%`
	`ENDFUNCTION`
	
	Produces :
	
	given: Expression
	wants: byte
	"TestDDGui.gbas"(5) error : wrong argument type : rep, arg no: 1
	
	because we are passing a string into a byte variable

command not inside function or sub	This occurs when trying to use a command (usually something like LOCAL) that can either be used only in the main program file or in a function, outside either of these limitations – for example, using LOCAL outside a function in a source file.
a sub/fkt is defined twice	A function or subroutine is defined twice
endselect without select	An ENDSELECT statement doesn't have a corresponding SELECT statement in the same code block
SELECT without ENDSELECT	A SELECT statement doesn't have a corresponding ENDSELECT statement in the same code block
ENDIF without IF	An ENDIF statement doesn't have a corresponding IF statement in the same code block
IF without ENDIF / THEN	An IF statement doesn't have a corresponding ENDIF (or THEN) statement in the same code block
ELSE without IF	An ELSE statement doesn't have a corresponding IF statement in the same code block
syntax error	A general typing error, like trying to assign a value to another value (eg 1=TRUE)
make use of protected keyword	Trying to assign a value to a reserved word
no valid license for this feature warning : Demo mode: This command would require "GLBasic SDK premium" :	The first version of the message is not used – the second version is used instead
cannot open a file?	Hasn't been seen, but most likely will be produced if a temporary file is deleted or can't be opened for some reason
BREAK outside of FOR/WHILE	BREAK can only be used within FOR/NEXT, WHILE/END and REPEAT/UNTIL statements – any use elsewhere is invalid
CONTINUE outside of FOR/WHILE	CONTINUE can only be used within FOR/NEXT, WHILE/END and REPEAT/UNTIL statements – any use elsewhere is invalid
redefinition as different type	Error is defined, but hasn't been seen yet
command must be outside function or sub	A function is inside another function. This is not allowed.
missing DATASTART for label	Whilst DATA statement errors produce a syntax error, this one may be produced if a label is

	incorrect
DIM has too many dimensions	When using a DIM statement, the variable being defined can only have up to 4 indices – any more and this error is generated.
requires standard parameter for	Error is defined, but hasn't been seen yet
type member is defined twice	A variable in a TYPE is defined more than once
type member is defined recursively	A variable in a TYPE is defined as the type. GLBasic does not allow recursive types. An example of this error is : TYPE t a AS t ENDTYPE
user type is not defined	A variable is being defined using a type that doesn't exist
variable is not explicitly defined/ implicitly created GLOBAL	When “Explicit Variable Declaration” is activated, this error message is displayed when you assign a value to a variable that hasn't been previously defined. In debug mode, if a variable isn't defined (and Explicit Variable Declaration isn't active), then the second message will be defined.
CASE without SELECT	A CASE statement doesn't have a corresponding SELECT statement in the same code block
TRY, CATCH, FINALLY mismatch	When using TRY, CATCH and FINALLY, they all must be used in the order : TRY … CATCH error$ … FINALLY
preprocessor error/preprocessor warning	Usually created when there is a problem either using a C file or INLINE code
No file	A file was left open when a program ends. This is usually an INI file
... was not declared in this scope	A type is being used that doesn't exist
called with 1 args. Required: x to x	This message is displayed when you call a function with too many or too few parameters.

Variable already exists	Redefining the same variable more than once
probably unassigned variable	A variable is defined but not used

Compiler Error Messages

Compiler Error Message :	Meaning :
invalid preprocessing directive	A preprocessing C command was used incorrectly
assignment of read-only variable `__GLBASIC__::xxxx	A read-only variable (defined by CONSTANT) is trying to be modified
multiple definition of	A variable is trying to be defined more than once
no matching function for call to	A call to a function doesn't exist
bug in IsArrayDecl. String was:	This only occurs if a calculation within an array is very complex (calling multiple functions, for example)
ISO C++ forbids taking the address of an unqualified or parenthesized non-static member function to form a pointer to member function	This message occurs when you are using SORTARRAY with a function in an extended type. The sorting function needs to be outside it.
error: non-lvalue in assignment	A function is being assigned a value, eg : `func()=1`
expected unqualified-id before '=' token	Trying to assign a value to a reserved word
error: ISO C++ forbids declaration of `a' with no type	A variable inside a type is the same as name of the type
error: conflicting declaration 'DGInt __GLBASIC__::	A label has the same name as a variable

Android Compiler Messages

Error Message :	Meaning :
AndroidManifest.xml:8: error: Error: No resource found that matches the given name (at 'icon' with value '@drawable/icon')	The androidicon.png file is missing for the project directory (not the .app directory!). Copy into there, perform a “clean project” operation and then compile again.
271: exec returned: 1	Denotes no actual machine or emulator connected or running to the machine when project is being uploaded. This message can be ignored.

Available commands in GLBasic

?

?DEFINE	-	Define a preprocessor variable
?ELSE	-	Perform actions if a ?IF comparison failed
?ENDIF	-	Terminate ?IF block
?ERROR	-	Generate an error message and stop compiling
?IF	-	Start a conditional block
?IF DEFINED	-	Check to see if one or more variables has been defined
?IFDEF	-	Check to see if a variable has been defined
?IFNDEF	-	Check to see if a variable has not been defined
?UNDEF	-	Un-define a preprocessor variable
?WARNING	-	Generate a warning message, but continue compiling

A

ABS	-	Return absolute value of a number
ACOS	-	Return angle of a value
ADDRESSOF	-	Return address of a function
ALIAS	-	Create an alias of a variable
ALLOWESCAPE	-	Enable or disable the ESC key
ALPHAMODE	-	Set the alpha value for graphics drawing
ALPHATESTING	-	Change transparency range
AND	-	Return result of logical AND
ANIMCOLL	-	Perform collision detection on animated sprites
AS	-	Define variable as a type
ASC	-	Convert a character to an ASCII value
ASIN	-	Returns an angle of a value
ASL	-	Returns a value shifted left
ASR	-	Returns a value shifted right
ASSERT	-	Generates an assertion error based on a result
ATAN	-	Returns the angle of a value
AUTOPAUSE	-	Enable or disable auto-pausing

B

BLACKSCREEN - De-activate LOADBMP and USEASBMP
BLENDSCREEN - Smooth fade a screen
BOUNDS - Return number of entries in an array
BOXCOLL - Perform collision detection between rectangles
BREAK - Terminate current loop
BYREF - Pass a variable by reference

C

CALLBACK - Allow a function to be overridden by another function
CALLBYNAME - Call a subroutine by name
CASE - Select action inside SELECT statement
CATCH - Catch a message
CHR$ - Convert ASCII number to character
CLEARERROR - Reset value of GETLASTERROR$()
CLEARSCREEN - Set colour of background buffer
CLOSEFILE - Close a file
CODEFILE$ - Return current GBAS file
CODELINE - Return current GBAS line number
CONSTANT - Define a variable that can't be modified
CONTINUE - Start next iteration of loop
COPYFILE - Copy one file to another
COS - Return the cosine of a value
CREATEDIR - Create a directory
CREATESCREEN - Create a virtual screen

D

DATA - Define a block of data
DEBUG - Write text to the debug (output) window in the IDE
DEC - Decrease the value of a variable
DECLARE - Load a DLL file with stdcall calling convention
DECLARE_ALIAS - Load a DLL file with stdcall calling mangled convention
DECLARE_C - Load a DLL file with cdecl calling convention
DECLARE_C_ALIAS - Load a DLL file with cdecl calling mangled convention
DECRYPT$ - Decrypt an encoded string
DEFAULT - Call code if previous CASE statements aren't called
DELETE - Delete an item from an array
DIM - Create a fixed size variable array

D Cont

DIMDATA	-	Initialise an array with values
DIMDEL	-	Delete an item from an array
DIMPUSH	-	Add an item to an array
DLLCALL	-	Manually load a DLL
DOESDIREXIST	-	Find out whether a directory exists or not
DOESFILEEXIST	-	Find out whether a file exists or not
DRAWANIM	-	Draw an animated sprite
DRAWLINE	-	Draw a line
DRAWRECT	-	Draw a rectangle
DRAWSPRITE	-	Draw a sprite

E

ELSE	-	Provide alternative if IF statement fails
ELSEIF	-	Combines ELSE and IF into one statement
ENCRYPT$	-	Encrypt a string
END	-	End the program
ENDDATA	-	End a data block
ENDFUNCTION	-	Denotes the end of a function
ENDIF	-	Denotes the termination of an IF block
ENDINLINE	-	Ends the ability to add C/C++ code
ENDOFFILE	-	Check to see if file pointer is at the end of a file
ENDPOLY	-	Ends a STARTPOLY section
ENDSELECT	-	Terminates a SELECT statement
ENDSUB	-	Denotes the end of a subroutine
ENDTYPE	-	Denotes the end of a user-defined structure
EXPORT	-	Export a function in a DLL

F

FALSE	-	Holds the value of 0
FILEPOSITION	-	Returns file position within a file
FILEREQUEST$	-	Display a file dialogue window
FILESEEK	-	Change the file position pointer in a file
FINALLY	-	Closes a TRY block
FINDPATH	-	Search for a path between points
FMOD	-	Return the remainder of a division
FOR	-	Initialise a FOR/NEXT loop
FORCEFEEDBACK	-	Activate force feedback on a compatible joystick
FOREACH	-	Iterate through an array

F Cont

FORMAT$	-	Formats a number
FUNCTION	-	Define a subroutine that can take and return values

G

GENFILE	-	Find first vacant file handle
GENFONT	-	Find first vacant font index
GENSOUND	-	Find first vacant sound index
GENSPRITE	-	Find first vacant sprite index
GENX_OBJ	-	Find first vacant 3D object index
GETCOMMANDLINE$	-	Retrieve command line parameters
GETCURRENTDIR$	-	Retrieve current directory
GETDESKTOPSIZE	-	Retrieve the width and height of the desktop
GETDIGIX	-	Retrieve X joystick HAT value
GETDIGIY	-	Retrieve Y joystick HAT value
GETENV$	-	Get value of an environment variable
GETFILE	-	Read a line from a file
GETFILELIST	-	Get list of files in current directory
GETFILESIZE	-	Get size of a file
GETFONTSIZE	-	Get font size of current font
GETJOYBUTTON	-	Retrieve button state
GETJOYNAME$	-	Get the name of a specific joystick
GETJOYRX	-	Get right X joystick value
GETJOYRY	-	Get right Y joystick value
GETJOYRZ	-	Get right Z joystick value
GETJOYX	-	Get X joystick value
GETJOYY	-	Get Y joystick value
GETJOYZ	-	Get Z joystick value
GETLASTERROR$	-	Get last error number and description
GETMOUSECOUNT	-	Get the number of attached mouse devices
GETNUMJOYSTICKS	-	Get the number of available joysticks
GETORIENTATION	-	Get display orientation
GETPIXEL	-	Read the colour of a pixel
GETSCREENSIZE	-	Retrieve screen resolution size
GETSPRITESIZE	-	Retrieve size of a sprite
GETTIMER	-	Return time since last call to SHOWSCREEN
GETTIMERALL	-	Return time since program started
GETVIEWPORT	-	Get viewport size
GLOBAL	-	Define a global variable
GOSUB	-	Call subroutine and then continue execution afterwards
GOTO	-	Jump to a specific label

G Cont

GRABSPRITE - Create a sprite from the back-buffer

H

HIBERNATE - Program sleeps until user action performed
HUSH - Stops all sounds except for music

I

IF - Evaluate a condition and control program flow
IMPORT - Make external functions available for use
IN - Define which list in FOREACH command to use
INC - Increase the value of a variable
INIGET$ - Retrieve data from an INI file
INIOPEN - Open an initialisation (INI) file
INIPUT - Write data in an INI file
INKEY$ - Returns the key the user pressed as a character string
INLINE - Allow embedding of C/C++ commands and functions
INPUT - Wait for user input
INSTR - Find characters in a string
INTEGER - Convert a number into an integer
ISFULLSCREEN - Returns window status of graphics screen
ISMUSICPLAYING - Return status of music playing

J

JOYSTATE - Set state of joystick 0

K

KEY - Return status of key pressed
KEYWAIT - Wait until the user presses a key
KILLFILE - Delete a file

L

LCASE$	-	Convert letters to lower-case
LEFT$	-	Extract characters from the left side of a string
LEN	-	Return number of ASCII characters in a string
LET	-	Assign a value to a variable
LIMITFPS	-	Set the maximum frame rate
LOADANIM	-	Load an animated sprite
LOADBMP	-	Load a graphic to be used as a background
LOADBUMPTEXTURE	-	Load a bump texture file
LOADFONT	-	Load a font file
LOADSOUND	-	Load a WAV file
LOADSPRITE	-	Load a graphics file
LOADSPRITEMEM	-	Load a sprite into an array
LOCAL	-	Define a local variable
LOGN	-	Logarithm to a base
LOOPMOVIE	-	Loop a movie until the SPACEBAR is pressed
LTRIM$	-	Remove text from the left side of a string

M

MAX	-	Returns larger of two values
MEM2SPRITE	-	Create a sprite from data
MID$	-	Extract one or more characters from a string
MIN	-	Returns the smaller of two values
MOD	-	Returns the integer part of a division
MOUSEAXIS	-	Read information about mouse
MOUSESTATE	-	Get a mouse X, Y and button status
MOUSEWAIT	-	Wait until mouse button is pressed
MUSICVOLUME	-	Set volume of music

N

NETALLOWJOINING	-	Allow or disallow computers to join a server
NETCREATEPLAYER	-	Create a network player
NETDESTROYPLAYER	-	Destroy a players connection to a server
NETGETIP$	-	Get list of IP addresses used by the computer
NETGETLASTERROR$	-	Get last network error number and description
NETGETMSG$	-	Get network message from specific player
NETGETPLAYERID	-	Get ID of a specific player
NETGETQUITPLAYER	-	Get ID of player who has quit
NETGETSENDER	-	Get ID of player that last sent message

N Cont

NETHOSTGAME	-	Host a network game
NETISACTIVE	-	Determine whether network system is running
NETJOINGAME	-	Join a network game
NETNUMPLAYERS	-	Get number of players seen by the server
NETPLAYERNAME$	-	Return display name of a player
NETSENDMSG	-	Send a network message
NETSHUTDOWN	-	Shut down networking system
NETWEBEND	-	End the program and launcher web browser
NETWEBGET	-	Retrieve contents of URL and store in a file
NETWEBGET$	-	Retrieve contents of URL and store in a string
NEXT	-	Denote end of FOR loop
NOT	-	Invert the result of a logic expression

O

ON_DEBUG	-	Interact with the debugger (for internal use only)
OPENFILE	-	Open a file for reading, writing or appending
OR	-	Perform a logical OR operation

P

PAUSEMUSIC	-	Pause playing music
PLATFORMINFO$	-	Retrieve platform information
PLAYMOVIE	-	Plays a movie
PLAYMUSIC	-	Plays a music file
PLAYSOUND	-	Plays a sound
POLYNEWSTRIP	-	Insert a de-generated triangle
POLYVECTOR	-	Define a polygon
POW	-	Returns value raised to a power
PRINT	-	Puts text onto the back-buffer
PROTOTYPE	-	Get function address
PUTENV	-	Set the value of an environment variable
PUTFILE	-	Write data to a line in a file

Q

- No commands -

R

READ - Read an item of data
READBYTE - Read a byte from an open file
READIEEE - Read a double precision float from an open file
READLINE - Read a line of text
READLONG - Read a long from an open file
READSHORTIEEE - Read a single precision float from an open file
READSTR - Read a given number of characters from an open file
READUBYTE - Read an unsigned byte from an open file
READULONG - Read an unsigned long from an open file
READUWORD - Read an unsigned word from an open file
READWORD - Reads a signed word from an opened file
REDIM - Resize an array
REPEAT - Start a loop that will run at least once
REPLACE$ - Replace text
REQUIRE - Include a C/C++ source file into the program
RESTORE - Define which block of data to read from
RETURN - Return a value back from a subroutine or function
REVINSTR - Reverse search in a string
RGB - Create one value containing red, blue and green values
RIGHT$ - Extract characters from the right-side of a string
RND - Generate a pseudo-random number
ROTOSPRITE - Draw a rotated sprite
ROTOZOOMANIM - Draw a rotated and scaled animated sprite
ROTOZOOMSPRITE - Draws a rotated and scaled sprite
RTRIM$ - Trim text from the right side of text

S

SAVEBMP - Saves the back buffer to a file
SAVESPRITE - Save a sprite to file
SEEDRND - Initialise random number generator with seed value
SELECT - Start complex conditional branching
SETACTIVEMOUSE - Select mouse to receive input from
SETCURRENTDIR - Change current working directory
SETFONT - Use a loaded font to define how text looks
SETMOUSE - Set mouse X and Y positions
SETPIXEL - Puts a coloured pixel onto the back buffer
SETSCREEN - Set screen resolution to a given size
SETSHOEBOX - Specific which files to use for protected data
SETSPRITEANIM - Sets tile size for sprites in DRAWANIM
SETTRANSPARENCY - Set transparency colour

S Cont

SGN	-	Return sign of a number
SHELLCMD	-	Execute another program or call DOS/Terminal
SHELLEND	-	Terminates current program and calls another
SHOWSCREEN	-	Swap front and back buffer
SETORIENTATION	-	Set display orientation
SIN	-	Returns the sine of a number
SLEEP	-	Halt program for a specific amount of time
SMOOTHSHADING	-	Set type of interpolation for scaling sprites
SOCK_CLOSE	-	Close a socket
SOCK_GETIP	-	Convert IP address string to an integer
SOCK_GETIP$	-	Convert IP address to readable string
SOCK_GETREMOTEIP	-	Get IP address of connected computer
SOCK_GETREMOTEPORT	-	Get port number of connected computer
SOCK_INIT	-	Initialise the socket system
SOCK_RECV	-	Receive a TCP or UDP message
SOCK_SETBLOCKING	-	Change socket blocking status
SOCK_SHUTDOWN	-	Shuts down the socket system
SOCK_TCPACCEPT	-	Return socket if computer connected to a port
SOCK_TCPCONNECT	-	Connect to a TCP server
SOCK_TCPLISTEN	-	Listen for a TCP message
SOCK_TCPSEND	-	Send a TCP message
SOCK_UDPOPEN	-	Open a UDP socket
SOCK_UDPSEND	-	Send a message
SORTARRAY	-	Sort an array
SOUNDPLAYING	-	Determines whether a sound is being played
SPLITSTR	-	Splits contents of a string into separate sections
SPRCOLL	-	Performs collision detection between sprites
SPRITE2MEM	-	Put sprite graphic data in array
SQR	-	Calculate the square root of a value
STARTDATA	-	Starts a data block
STARTPOLY	-	Allow the drawing of polygons
STATIC	-	Define a static variable
STDCOLOR	-	Sets the colour for the next STDOUT command
STDERR	-	Sends text to standard error stream
STDIN$	-	Read text from standard input stream
STDOUT	-	Send text to standard output stream
STEP	-	Increases FOR counter by a defined amount
STOPMUSIC	-	Stops music being played
STOPSOUND	-	Stops a sound being played
STRETCHANIM	-	Draws and resizes an animated sprite
STRETCHSPRITE	-	Draws and resizes a sprite
SUB	-	Define a subroutine

S Cont

SWAP - Swap the value of two variables
SYSTEMPOINTER - Enable or disable the system mouse pointer

T

TAN - Returns the tangent of a value
THEN - Define what action to take after IF statement
THROW - Generate an exception error
TO - Introduce terminating condition for FOR loop
TRIM$ - Trim text
TRUE - Holds a value of 1
TRY - Initialise the ability to catch errors
TYPE - Allow creation of groups

U

UCASE$ - Convert characters to upper-case
UNTIL - Evaluate condition for REPEAT loop
URLDECODE$ - Decode a URL string
URLENDCODE$ - Encode a URL string
USEASBMP - Use current background as background image
USESCREEN - Perform action on a virtual screen

V

VIEWPORT - Define area in which graphics are plotted

W

WEND - Denotes the end of a WHILE loop
WHILE - Execute code whilst a condition evaluates to **TRUE**
WRITEBYTE - Write a signed-byte to an open file
WRITEIEEE - Writes an eight byte floating-point value to an open file
WRITELINE - Writes a line of data to a file
WRITELONG - Writes a four byte integer to an open file
WRITESHORTIEEE - Write a four byte floating-point value to an open file
WRITESTR - Writes a given number of bytes to an open file
WRITEUBYTE - Writes an unsigned byte to an open file

W Cont

WRITEULONG - Writes an unsigned long to an open file
WRITEUWORD - Writes an unsigned word to an open file
WRITEWORD - Writes a signed word to an open file

X

X_AMBIENT_LT - Enables ambient lighting
X_AUTONORMALS - Choose mode for lighting
X_CAMERA - Create a camera
X_CAMERAUP - Set camera's up direction
X_CLEAR_Z - Clear the depth buffer
X_COLLISION - Performs a collision check between two objects
X_COLLISIONAABB - Perform bounding box collision check
X_COLLISIONRAY - Performs a ray intersection collision check
X_COLLISIONSPHEREBOX - Performs a cuboid collision detection against a sphere
X_CULLMODE - Determine how a 3D object is displayed
X_DOT - Draws a 3D dot
X_DRAWANIM - Draw an animation frame of a 3D object
X_DRAWAXES - Draws debugging axes
X_DRAWOBJ - Draw a 3D object
X_ENABLE_Z - Enable or disable Z buffer
X_FOG - Activate or deactivate fog
X_GETCAMERAMATRIX - Get 3D camera matrix
X_GETCOLLISIONFACE - Get index of surface that triggers a collision
X_GETFACE - Retrieve information about a triangle
X_GETMATRIX - Get transformation matrix
X_LINE - Draw a 3D line
X_LOADOBJ - Load a 3D object into memory
X_LOADSHADER - Load a GLSL shader
X_MAKE2D - Switch to 2D mode
X_MAKE3D - Switch to 3D mode
X_MIPMAPPING - Enable or disable mipmapping
X_MOVEMENT - Set the position of the next 3D object
X_MULTMATRIX - Access the OpenGL "*glMultiMatrix*" function
X_NUMFACES - Get the number of triangles in a 3D object
X_OBJADDVERTEX - Add a vertex to a 3D object being created
X_OBJEND - End creation of 3D objects
X_OBJNEWGROUP - Create a new group of vertices
X_OBJSETFRAME - Switch to another frame when creating objects
X_OBJSTART - Start creating a 3D object at runtime
X_POPMATRIX - De-group objects

X Cont

X_PRINT	-	Draw 3D text
X_PUSHMATRIX	-	Group objects
X_PUTSHADER	-	Allow changing of uniform float value
X_ROTATION	-	Sets the rotation angle for the next 3D object
X_SAVEOBJ	-	Save a 3D object to a file
X_SCALEMODEL	-	Scale a 3D object
X_SCALING	-	Set scaling size for the next 3D object
X_SCREEN2WORLD	-	Convert 2D coordinates to 3D
X_SETCELSHADES	-	Define brightness level for a shader
X_SETSHADER	-	Set the shader to be used
X_SETTEXTURE	-	Set texture to be used on a 3D object
X_SETTEXTUREOFFSET	-	Shift texture polygon
X_SPHEREMAPPING	-	Enable or disable sphere mapping
X_SPOT_LT	-	Enable spot lighting
X_SPRITE	-	Draws a sprite at 3D coordinates
X_WORLD2SCREEN	-	Convert 3D coordinates to 2D coordinates

Y

- There are no commands -

Z

ZOOMSPRITE	-	Draw a rescaled sprite

b

bAND	-	Return result of binary AND operation
bNOT	-	Return result of binary NOT operation
bOR	-	Return result of binary OR operation
bXOR	-	Return result of binary exclusive-OR operation

g

get_sprite_texture	-	Get OpenGL texture ID

S

`self` - Access variable or function in an extended type

Member Operator

dot (`.`) - Access member of a type

Comment Operator

`//` - Comment the line

Command Listing by Category

Pre-processor Commands

?DEFINE, ?ELSE, ?ENDIF, ?ERROR, ?IF, ?IFDEF, ?IFNDEF, ?UNDEF, ?WARNING

File Commands

CLOSEFILE, COPYFILE, CREATEDIR, DOESDIREXIST, DOESFILEEXIST, ENDOFFILE, FILEPOSITION, FILEREQUEST$, FILESEEK, GENFILE, GETCURRENTDIR$, GETFILE, GETFILELIST, GETFILESIZE, INIGET$, INIOPEN, INIPUT, KILLFILE, OPENFILE, PUTFILE, READBYTE, READIEEE, READLINE, READLONG, READSHORTIEEE, READSTR, READUBYTE, READULONG, READUWORD, READWORD, SETCURRENTDIR, WRITEBYTE, WRITEIEEE, WRITELINE, WRITELONG, WRITESHORTIEEE, WRITESTR, WRITEUBYTE, WRITEULONG, WRITEUWORD, WRITEWORD

Maths Commands

ABS, ACOS, ASIN, ASL, ASR, ATAN, COS, DEC, FMOD, INC, INTEGER, LOGN, MAX, MIN, MOD, POW, SGN, SIN, SQR, TAN, bAND, bNOT, bOR, bXOR

2D Commands

ALPHAMODE, ALPHATESTING, BLACKSCREEN, BLENDSCREEN, BOXCOLL, CLEARSCREEN, CREATESCREEN, DRAWLINE, DRAWRECT, ENDPOLY, GETDESKTOPSIZE, GETORIENTATION, GETPIXEL, GETSCREENSIZE, GETVIEWPORT, ISFULLSCREEN, LIMITFPS, LOADBMP, LOADBUMPTEXTURE, POLYNEWSTRIP, POLYVECTOR, RGB, SAVEBMP, SETORIENTATION, SETPIXEL, SETSCREEN, SETTRANSPARENCY, SHOWSCREEN, SMOOTHSHADING, STARTPOLY, SYSTEMPOINTER, USEASBMP, USESCREEN, VIEWPORT

Sprite Commands

ANIMCOLL, DRAWANIM, DRAWSPRITE, GENSPRITE, GETSPRITESIZE, GRABSPRITE, LOADANIM, LOADSPRITE, LOADSPRITEMEM, MEM2SPRITE, ROTOSPRITE, ROTOZOOMANIM, ROTOZOOMSPRITE, SAVESPRITE, SETSPRITEANIM, SPRCOLL, SPRITE2MEM, STRETCHANIM, STRETCHSPRITE, ZOOMSPRITE, get_sprite_texture

3D Commands

GENX_OBJ, X_AMBIENT_LT, X_AUTONORMALS, X_CAMERA, X_CAMERAUP, X_CLEAR_Z, X_COLLISION, X_COLLISIONAABB, X_COLLISIONRAY, X_COLLISIONSPHEREBOX, X_CULLMODE, X_DOT, X_DRAWANIM, X_DRAWAXES, X_DRAWOBJ, X_ENABLE_Z, X_FOG, X_GETCAMERAMATRIX, X_GETCOLLISIONFACE, X_GETFACE, X_GETMATRIX, X_LINE, X_LOADOBJ, X_LOADSHADER, X_MAKE2D, X_MAKE3D, X_MIPMAPPING, X_MOVEMENT, X_MULTMATRIX, X_NUMFACES, X_OBJADDVERTEX, X_OBJEND, X_OBJNEWGROUP, X_OBJSETFRAME, X_OBJSTART, X_POPMATRIX, X_PRINT, X_PUSHMATRIX, X_PUTSHADER, X_ROTATION, X_SAVEOBJ, X_SCALING, X_SCREEN2WORLD, X_SETCELSHADES, X_SETSHADER, X_SETTEXTURE, X_SETTEXTUREOFFSET, X_SPHEREMAPPING, X_SPOT_LT, X_SPRITE, X_WORLD2SCREEN

Networking Commands

NETALLOWJOINING, NETCREATEPLAYER, NETDESTROYPLAYER, NETGETIP$, NETGETLASTERROR$, NETGETMSG$, NETGETPLAYERID, NETGETQUITPLAYER, NETGETSENDER, NETHOSTGAME, NETISACTIVE, NETJOINGAME, NETNUMPLAYERS, NETPLAYERNAME$, NETSENDMSG, NETSHUTDOWN, NETWEBEND, NETWEBGET, NETWEBGET$, SOCK_CLOSE, SOCK_GETIP, SOCK_GETIP$, SOCK_GETREMOTEIP, SOCK_GETREMOTEPORT, SOCK_INIT, SOCK_RECV, SOCK_SETBLOCKING, SOCK_SHUTDOWN, SOCK_TCPACCEPT, SOCK_TCPCONNECT, SOCK_TCPLISTEN, SOCK_TCPSEND, SOCK_UDPOPEN, SOCK_UDPSEND

Font Commands

GENFONT, GETFONTSIZE, LOADFONT, SETFONT

Inline/External Code Commands

DECLARE, DECLARE_ALIAS, DECLARE_C, DECLARE_C_ALIAS, ENDINLINE, EXPORT, IMPORT, INLINE, REQUIRE

User Input Commands

FORCEFEEDBACK, GETJOYBUTTON, GETJOYNAME$, GETDIGIX, GETDIGIY, GETJOYRX, GETJOYRY, GETJOYRZ, GETJOYX, GETJOYY, GETJOYZ, GETMOUSECOUNT, GETNUMJOYSTICKS, INKEY$, INPUT, JOYSTATE, KEY, KEYWAIT, MOUSEAXIS, MOUSESTATE, MOUSEWAIT, SETACTIVEMOUSE, SETMOUSE

String Commands

ASC, CHR$, DECRYPT$, ENCRYPT$, FORMAT$, INSTR, LCASE$, LEFT$, LEN, LTRIM$, MID$, PRINT, REPLACE$, REVINSTR, RIGHT$, RTRIM$, SPLITSTR, TRIM$, UCASE$, URLDECODE$, URLENCODE$

Error Checking/Debugging Commands

ASSERT, CATCH, CLEARERROR, CODEFILE$, CODELINE, DEBUG, FINALLY, GETLASTERROR$, THROW, TRY,

Loop/Control Commands

BREAK, CALLBACK, CALLBYNAME, CASE, CONTINUE, DEFAULT, ELSE, ELSEIF, ENDFUNCTION, ENDIF, ENDSELECT, ENDSUB, FOR, FOREACH, FUNCTION, GOSUB, GOTO, IF, IN, NEXT, REPEAT, RETURN, SELECT, STEP, SUB, THEN, TO, UNTIL, WEND, WHILE

Logic Commands

AND, NOT, OR

Array Commands

ADDRESSOF, BOUNDS, DELETE, DIM, DIMDATA, DIMDEL, DIMPUSH, REDIM, SORTARRAY

Sound/Music/Video Commands

GENSOUND, HUSH, ISMUSICPLAYING, LOADSOUND, LOOPMOVIE, MUSICVOLUME, PLAYMOVIE, PLAYMUSIC, PLAYSOUND, SOUNDPLAYING, STOPMUSIC, STOPSOUND

Environment/Command Line Commands

GETCOMMANDLINE$, GETENV$, PUTENV, SHELLCMD, SHELLEND, STDCOLOR, STDERR, STDIN$, STDOUT

Data Commands

DATA, ENDDATA, READ, RESTORE, STARTDATA

Variable Definition/Manipulation Commands

ALIAS, AS, CONSTANT, END, ENDTYPE, FALSE, GLOBAL, LET, LOCAL, STATIC, SWAP, TRUE, TYPE

Program Execution Commands

ALLOWESCAPE, AUTOPAUSE, BYREF, FINDPATH, HIBERNATE, PLATFORMINFO$, PROTOTYPE, SLEEP

Time Commands

GETTIMER, GETTIMERALL

Random Number Commands

RND, SEEDRND

Compression Commands

SETSHOEBOX

?DEFINE

Type : Define pre-processor variable

Syntax : ?DEFINE <var> <value>

See Also : ?UNDEF

Notes :

- Currently, variables defined are only local to the file in which they have been created in, unless created in the Main program file, in which case they will be global.

Result : Defines the preprocessor variable **var** with the value held in **value**

Example :

```
// This example allows defines a preprocessor variable A, and sets it to 1
?DEFINE A 1
```

?ELSE

Type : Alternative path if pre-processor comparison failed

Syntax : ?ELSE

See Also : ?IF, ?ENDIF

Notes : None

Result : This command is needed if you want to run alternative code if the proceeding ?IF condition fails.

Example :

```
// This example will generate a warning message “A = 2” if the preprocessor
// variable is not equal to 1
?DEFINE A 2
?IF A == 1
  ?WARNING "A = 1"
?ELSE
  ?WARNING "A = 2"
?ENDIF
```

?ENDIF

Type : End pre-processor comparison block

Syntax : ?ENDIF

See Also : ?IF, ?ELSE

Notes : None

Result : Terminates the nearest ?IF block

Example :

```
// This example allows terminates a ?IF block
?IF A > 0
  // Do something
?ENDIF
```

?ERROR

Type : Display pre-processor error message

Syntax : ?ERROR *[<string>]*

See Also : ?WARNING

Notes : None

Result : This command displays “error : preprocessor error : “, followed by the contents of **string**, in the results window

Example :

```
// This example will generate error : preprocessor error : An error occurred
?ERROR “An error occurred”
```

?IF

Type : Pre-processor comparison command

Syntax : ?IF <variable> *[<comparisonSign> <value>]*

See Also : ?ELSE, ?ENDIF

Notes :

- If a variable is not defined, the compiler will stop with "error : syntax error :"

Result : This command starts a comparison block, by checking the value held in **variable**, against the comparison sign (defined by *comparisonSign*) and value (if *present*). If the comparison result is **TRUE**, then the following code is executed.

Example :

```
// This example checks A to see if equals 1, is greater than 2 and if less than 55
?IF A == 1
  ?WARNING "A = 1"
?ENDIF

?IF A > 2
  ?WARNING "A is greater than 2"
?ENDIF

?IF A<55
  ?WARNING "A is less than 55"
?ENDIF
```

?IF DEFINED

Type : Pre-processor comparison command

Syntax : ?IF DEFINED(<variable>) *[AND/OR DEFINED(<variable2>)]*

See Also : ?IFDEF

Notes : None

Result : This command checks to see if **variable** has been defined by an ?IF statement. If so, the following code is executed. Unlike ?IFDEF, multiple variables can be validated by using +, -, *, /, AND, OR, ( ) and DEFINED(variable)

Example :

```
// This example checks to see if A has been defined
?IF DEFINED(A)
  ?WARNING "A has been defined"
?ENDIF

?IF DEFINED(A) OR DEFINED(B)
  ?WARNING "A or B is defined"
?ENDIF
```

?IFDEF

Type : Pre-processor comparison command

Syntax : ?IFDEF <variable>

See Also : ?IFNDEF, ?ENDIF, ?ELSE, ?IF

Notes : None

Result : This command checks to see if **variable** has been defined by an ?IF statement. If so, the following code is executed

Example :

```
// This example checks to see if A has been defined
?IFDEF A
  ?WARNING "A has been defined"
?ENDIF
```

?IFNDEF

Type : Pre-processor comparison command

Syntax : ?IFNDEF <variable>

See Also : ?IF, ?IFDEF, ?ELSE, ?ENDIF

Notes : None

Result : This command checks to see if **variable** has not been defined by an ?IF statement. If so, the following code is executed

Example :

```
// This example checks to see if A has not been defined
?IFNDEF A
  ?WARNING "A has NOT been defined"
?ENDIF
```

?UNDEF

Type : Un-define pre-processor variable

Syntax : ?UNDEF <variable>

See Also : ?DEFINE

Notes : None

Result : This command un-defines the preprocessor variable **variable**. Subsequent references to **variable** are ignored unless it is defined again

Example :

```
// This example un-defines the variable A
?UNDEF A
```

?WARNING

Type : Display pre-processor warning message

Syntax : ?WARNING *[<string>]*

See Also : ?ERROR

Notes : None

Result : This command displays "warning : preprocessor warning : ", followed by the contents of **string**, in the results window

Example :

```
// This example will generate a warning message : preprocessor warning : This is a warning
?ERROR "This is a warning"
```

ABS

Type : Get absolute value of a number

Syntax : `result = ABS(<value>)`

See Also : SGN

Notes : None

Result : Returns the absolute value held in **value**.

Example :

```
// This example sets A to equal 12
A = ABS(-12.0)
```

ACOS

Type : Get angle of a value

Syntax : `result = ACOS(<value>)`

See Also : ASIN, ATAN, COS, SIN, TAN

Notes : None

Result : Returns the angle held in **value**

Example :

```
// This sets A to 89.42703266
A = ACOS(0.01)
```

ADDRESSOF

Type : SORTARRAY helper function

Syntax : `result = ADDRESSOF(<functionName>)`

See Also : SORTARRAY

Notes :

- This command should only be used with SORTARRAY
- The function CANNOT be within a type

Result : Returns the address of the function defined by **functionName**

Example :

```
// This example returns the address of the function called 'routine'
result=ADDRESSOF(routine)
```

ALIAS

Type : Pointer to variable

Syntax : `ALIAS <name> AS <variable>`

See Also : None

Notes :

- If the size of any involved arrays gets changed (using commands like `DIMPUSH`, `DIMDEL` and `DELETE`), this pointer will almost certainly be broken and your program will crash ...if you're lucky.

Result : Creates the alias of **name**, with the properties of **variable**.

An alias is a user-defined name which can be used in place of accessing a specific array index or type, reducing the amount of typing especially of complex variables

Example :

```
LOCAL thisIsALongVariable

// This alias allows the variable called W to be used in place of
// thisIsALongVariable
ALIAS W AS thisIsALongVariable

// This would set thisIsALongVariable to equal 3
W = 3
```

ALLOWESCAPE

Type : Control ESC key

Syntax : `ALLOWESCAPE <value>`

See Also : None

Notes : None

Result : Enables the ESC key if **value** = **TRUE**, and disables it if **value** = **FALSE**

Example :

```
// This enables the user to exit the program by pressing the ESC key
ALLOWESCAPE TRUE
```

ALPHAMODE

Type : Set graphic blending value

Syntax : ALPHAMODE <alpha>

See Also : ALPHATESTING

Notes : None

Result : Sets the alpha mode for all following drawing commands. The alpha level is set by alpha, and should be in the range of -1.0 to +1.0, and performs the following effects :

Value of **alpha**	Effect
< 0	Bright alpha blending
0	No alpha blending
> 0	Interpolated alpha blending

Example :

```
// This example removes alpha blending for all subsequent drawing commands
ALPHAMODE 0.0
```

ALPHTESTING

Type : Change transparency range

Syntax : ALPHATESTING <alpha>

See Also : ALPHAMODE

Notes :

- If **alpha** = 1.0, then all transparency is ignored (possibly giving a speed boost)

Result : This command allows the transparency colour strength to be changed, allowing a different range to be treated as a transparency colour.

alpha is the transparency range and should be between 0.0 and 1.0

Example :

```
// Sprite with hole
SETTRANSPARENCY RGB(255,0,128)
DRAWRECT 0,0,500,500,RGB(255,0,128)

ALPHAMODE -0.1
DRAWRECT 0,0,500,500,RGB(255,0,0)
ALPHAMODE 0
DRAWRECT 0,0,29,29,RGB(255,255,255)
DRAWRECT 1,1,27,27,RGB(0,0,255)

DRAWRECT 5,5,8,8,RGB(255,0,128) // hole
GRABSPRITE 0, 0,0, 32,32

CLEARSCREEN RGB(80,200,200) // Blue background

WHILE TRUE
   SMOOTHSHADING TRUE
   LOCAL phi = GETTIMERALL() / 30
   FOR i% = 0 TO 3
      x =100 + i*150

      SELECT i
         CASE 0 // linear interpolation of pixels
            SMOOTHSHADING FALSE
            ALPHAMODE -1.0
         CASE 1 // blury edges (default)
            SMOOTHSHADING TRUE
            ALPHATESTING 0.05
            ALPHAMODE -1.0
         CASE 2 // sharp edges
            SMOOTHSHADING TRUE
            ALPHATESTING 0.99
            ALPHAMODE -1.0
         CASE 3 // no transparency
            ALPHAMODE 0
            SMOOTHSHADING TRUE
            ALPHATESTING 1.0
      ENDSELECT

      ROTOZOOMSPRITE 0, x,100,phi, 3.5
   NEXT

   SHOWSCREEN
WEND
```

AND

Type : Logical operator

Syntax : `result = <value1> AND <value2>`

See Also : `OR, NOT`

Notes : None

Result : Returns **TRUE** if both **value1** AND **value2** are non-zero

Example :

```
// A is set to TRUE if both B and C are non-zero
A = B AND C
```

ANIMCOLL

Type : Sprite collision function

Syntax : `result = ANIMCOLL(<index1>, <tile1>, <x1>, <y1>, <index2>, <tile2>, <x2>, <y2>)`

See Also : SPRCOLL

Notes :

- If **index1** or **index2** are NOT animated sprites, then **tile1** or **tile2** should be set to 0

Result : Performs a per-pixel collision between two animated sprites.

The sprite numbers are defined by **index1** and **index2**, and the tile offset into these numbers are defined by **tile1** and **tile2**.

The positions of the objects to be compared are defined by **x1**, **y1** and **x2**, **y2**.

If a collision is detected, then this routine returns **TRUE**. If no collision is detected, then **FALSE** is returned.

Example :

```
// This example checks for collision between frame 0 of sprite index 0 at position 100, 100 and
// frame 0 of sprite index 16 at 110, 120
A = ANIMCOLL(0,0, 100, 100, 16, 0, 110, 120)
```

AS

Type : Type name definer

Syntax : `<var> AS <type>`

See Also : TYPE

Notes : None

Result : Defines the variable **var** as a type defined by **type**

Example :

```
// This example defines the variable one as the type called test
LOCAL one as test
```

ASC

Type : Character to number converter function

Syntax : `result% = ASC(<character$> [,index%=0])`

See Also : CHR$

Notes :

- From V10.179, an optional second parameter has been introduced to facilitate easy traversing of strings

Result : Converts one character stored in **character$** into it's equivalent ASCII code

Examples :

```
// This stores the ASCII character for A (65) into the variable A
A = ASC("A")

// Print the ASCII value of the first letter in the string Name$
GLOBAL Name$ = "Elvis"
PRINT ASC(Name$), 100, 100
SHOWSCREEN
MOUSEWAIT
OUTPUT: 69

// Print the ASCII value of the last letter in a string
GLOBAL Name$ = "Elvis"
PRINT ASC(Name$,len(Name$)-1), 100, 100
SHOWSCREEN
MOUSEWAIT
```

ASIN

Type : Get angle of a value

Syntax : `result = ASIN(<value>)`

See Also : ACOS, ATAN, COS, SIN, TAN

Notes : None

Result : Returns the angle held in **value**

Example :

```
// This sets A to 90
A = ASIN(13.9)
```

ASL

Type : Shift a value left

Syntax : `result% = ASL(<value>,<shift>)`

See Also : ASR

Notes : None

Result : Returns the value of **value** shifted left by the value in **shift**. This is equivalent of multiplying **value** by 2 to the power of **shift**

Example :

```
// This sets A to 16
A = ASL(1, 4)
```

ASR

Type : Shift a value right

Syntax : `result% = ASR(<value>,<shift>)`

See Also : ASL

Notes : None

Result : Returns the value of **value** shifted right by the value in **shift**. This is equivalent of dividing **value** by 2 to the power of **shift**

Example :

```
// This sets A to 8
A = ASR(128, 4)
```

ASSERT

Type : Debugging function

Syntax : `ASSERT <expression>`

See Also : None

Notes :

- An assertion error will immediately end the program
- This only works in debug mode
- If the GLB_ON_QUIT subroutine is defined, then the contents of it will be called.

Result : Generates an assertion error message (Error: Assertion failed), if **expression** evaluates to **FALSE**. The result of **expression** is displayed in the debugger window

Example :

```
// This example generates an assertion error, and displays in the debugger window :
// Assertion: Value = 0
// Error:
// Assertion failed
ASSERT 3=1 // Three does not equal one...

// This more complex example generates an assertion error when A% reaches 5.
// At this point the program will end
FOR A%=1 TO 10
ASSERT A%<5
NEXT
```

ATAN

Type : Return able of two values

Syntax : `result = ATAN(<x>, <y>)`

See Also : `ASIN, ACOS, COS, SIN, TAN`

Notes : None

Result : Returns the angle by calculating the ATAN of **x**/**y**

Example :

```
// Sets A to be 76.98119757
A = ATAN(17.3, 4.0)
```

AUTOPAUSE

Type : Define state of auto-pausing

Syntax : AUTOPAUSE <value>

See Also : None

Notes :

- This command should be used after a SETSCREEN call

Result : Enables auto-pausing if **value** is **TRUE**, and disables it if **value** is **FALSE**

From V8 onwards, if **value** is **TRUE**, then **GLB_ON_PAUSE** and **GLB_ON_RESUME** are called when a program loses focus (minimised or task switched to another program) or regains it

Example :

```
// This example will change the isPaused% variable depending on whether the program is minimised or
// not.  If it is, then a pause message is displayed
// once the program can continue
GLOBAL isPaused%=FALSE
LOCAL a%

AUTOPAUSE TRUE
WHILE TRUE
  IF isPaused%
    PRINT "This program is paused.  Press a mouse button to continue",0,0
    SHOWSCREEN
    MOUSEWAIT
  ELSE
    PRINT "MAIN LOOP:"+a%,0,0
    SHOWSCREEN
    HIBERNATE
    INC a%
  ENDIF
 WEND

 SUB GLB_ON_PAUSE:
   isPaused%=TRUE
 ENDSUB

 SUB GLB_ON_RESUME:
   isPaused%=FALSE
 ENDSUB
```

BLACKSCREEN

Type : Set colour of background screen

Syntax : BLACKSCREEN

See Also : LOADBMP, USEASBMP, SHOWSCREEN, CLEARSCREEN

Notes :
- This command is depreciated, and will be noted as such during compilation

Result : This command deactivates the use of LOADBMP and USEASBMP to produce a black screen at every call to SHOWSCREEN

Example :

```
// Use a black screen as a background
BLACKSCREEN
```

BLENDSCREEN

Type : Blend between two screens

Syntax : BLENDSCREEN <fileName$> [,<*duration*>]

See Also : None

Notes :
- On the iPhone, this command will only work in Portrait mode

Result : This command fades from the current contents of the current back-buffer to the graphic held in **fileName$**.

If ***duration*** is given, then the value held there will determine how the blending lasts. If no value is given, then the default value of 1000 ms is used

Example :

```
// Blend current screen with the graphic held in the file, lasting 1000 ms
BLENDSCREEN "BLEND.BMP"
```

BOUNDS

Type : Return array size

Syntax : `result% = BOUNDS(<array[]>,<index>)`

See Also : `LEN`

Notes : None

Result : Returns the number of entries in the array defined by **array**, and at the index defined by **index**

To retrieve the number for **index** 0, a slightly quicker way would be to use `LEN`(array[])

Example :

```
// Using the array...
DIM ARRAY[5][7]

// This example would return 5, as that is the maximum number of entries for index 0 in the
// ARRAY
DEBUG BOUNDS(ARRAY[],0)

// An alternative would be
DEBUG LEN(ARRAY[])

// This example returns 7, which is the maximum number of entries allowed in the second
// dimension in the array
DEBUG BOUNDS(ARRAY[],1)
```

BOXCOLL

Type : Box collision function

Syntax :
```
result = BOXCOLL(<x1>, <y1>, <width1>, <height1>,
                 <x2>, <y2>, <width2>, <height2>)
```

See Also : SPRCOLL

Notes : None

Result : Performs a box collision check between two rectangles. The X and Y coordinates of each is defined by **x1**, **y1** and **x2**, **y2** with a width and height of **width1**, **height1** and **width2** and **height2**.

If these two rectangles overlap, this command returns **TRUE**, otherwise **FALSE** is returned.

Example :

```
// This example checks to see if a box at position 100,100 with a width and height of 32 overlaps
// with a rectangle positioned at 80,70 width a width of
// 120 and a height of 60
A = SPRCOLL(100, 100, 32, 32, 80, 70, 120, 60)
```

BREAK

Type : Loop terminator

Syntax : BREAK

See Also : FOR, NEXT, WHILE

Notes :

- This command can also be used to break out of REPEAT/UNTIL loop

Result : Terminates the current loop statement (FOR, REPEAT or WHILE) immediately and transfers control to the statement immediately following the end of the loop

Example :

```
// Standard command use
BREAK

FOR X = 1 TO 10
   // If X = 1 then break out of the loop and continue executing the code after the NEXT statement
   IF X = 1
      BREAK
   ENDIF
NEXT

// Break out of a REPEAT/UNTIL loop
LOCAL a%=0

REPEAT
       IF a%>=100 THEN BREAK
       INC a%
UNTIL FALSE
```

BYREF

Type : Pass variable by reference

Syntax : `BYREF <var>`

See Also : None

Notes :

- This command should NOT be used with a TYPE - they are automatically passed by reference, and including this would generate compiler errors.

Result : `BYREF` allows a variable to be passed by reference. This means an implicit reference to **var** is passed, rather than a copy of its value, allowing **var** to be modified and with the new value seen by its caller

Example :

```
// This example allows the variable name$ to  be changed and the new value accessed by the callee
// of the function
FUNCTION test: BYREF name$
   name$ = "TEST"
ENDFUNCTION
```

CALLBACK

Type : Function modifier

Syntax : CALLBACK FUNCTION <function>: *[<var1>, <var2>, ...]*

See Also : FUNCTION

Notes :
- CALLBACK should only be used in included files
- Callback functions can't use % to return integer values

Result : CALLBACK allows the duplication of functions, which included files can then call.

Example :

```
// This example allows the function called test to be called in an included file and also in the main
// file
CALLBACK FUNCTION test:
  ...
ENDFUNCTION
```

CALLBYNAME

Type : Call a subroutine

Syntax : result% = CALLBYNAME(<name$>)

See Also : SUB

Notes : None

Result : The subroutine is called by the name held in **name$**. If the subroutine exists, then CALLBYNAME returns **TRUE**, otherwise it returns **FALSE**

Example :

```
// This example calls the subroutine called 'test'.  Result would hold TRUE as the subroutine exists
result = CALLBYNAME("test")

SUB test:
  ...
ENDSUB
```

CASE

Type : Option selector

Syntax : CASE [>]<var1> [*TO <var2>*]

See Also : SELECT, ENDSELECT, DEFAULT

Notes : None

Result : Evaluates the variable **var** or **var1** to **var2**, to determine whether the code following the CASE statement is executed

Example :

```
SELECT A
    CASE 7
          // Display 'Value is 7' if A = 7
          DEBUG "Value is 7"
ENDSELECT
```

CATCH

Type : Exception handler

Syntax : CATCH <error$>

See Also : TRY, THROW, FINALLY

Notes :

- This command MUST be included when using TRY and FINALLY

Result : Any messages generated by the THROW command will be processed here. The message will be stored in **error$**

Example :

```
// The general usage for CATCH is :
CATCH VAR$

// The general usage for CATCH is:
TRY
  // Code to be checked
  CATCH ERROR$
    // Process any errors here
    DEBUG "An error as occurred  The message is : "+ERROR$
FINALLY
```

CHR$

Type : ASCII to Character converter

Syntax : result$ = CHR$(<character%>)

See Also : ASC

Notes : None

Result : This function converts the character code held by **character%** into an associated string character.

Example :

```
// This example stores the ASCII character 65 (which is an A) into the variable A$
A$ = CHR$(65)
```

CLEARERROR

Type : Reset error result

Syntax : CLEARERROR

See Also : GETLASTERROR$

Notes : None

Result : Resets the result that would be returned from GETLASTERROR$

Example :

```
// This clears the GETLASTERROR$() return value
CLEARERROR

// This would return a value of  0 success
DEBUG GETLASTERROR$()
```

CLEARSCEEN

Type : Backbuffer graphics command

Syntax : CLEARSCREEN [*<colour%>=0*]

See Also : BLACKSCREEN

Notes :

- This command should be used instead of BLACKSCREEN

Result : The background colour is set to the value defined by **colour%**. Usually the RGB command is used to define the colour.

This colour is retained through the life of the program, or until this command is called again with a different colour.

If **colour%** = -1, then the background buffer isn't cleared, retaining all previously plotted graphics.

Example :

```
// This example sets the background buffer to red
CLEARSCREEN RGB(255,0,0)
WHILE TRUE
   SHOWSCREEN // A red screen is displayed
WEND

// This example retains all previous graphic plotting, and produces a psychedelic effect
CLEARSCREEN -1

WHILE TRUE
   DRAWRECT RND(200),RND(200),RND(200),RND(200),RGB(RND(255),RND(255),RND(255))
   SHOWSCREEN
WEND
```

CLOSEFILE

Type : File handling command

Syntax : `CLOSEFILE <handle%>`

See Also : `OPENFILE`

Notes : None

Result : Closes the file handle defined by **handle%**

Example :

```
// This closes the file with the handle of 1
CLOSEFILE 1
```

CODEFILE$

Type : Debugging function

Syntax : `result$ = CODEFILE$()`

See Also : `CODELINE`

Notes : None

Result : Returns the GBAS filename that the `CODEFILE$` command appears in

Example :

```
// This example will return the name of the GBAS file that the example was saved as.  If this
// command is used in multiple files, it will give the filename of each file it appears in
 DEBUG CODEFILE$()
```

CODELINE

Type : Debugging function

Syntax : `result% = CODELINE()`

See Also : `CODEFILE$`

Notes : None

Result : Returns the line number that the `CODELINE` command appears on.

Example :

```
// This example returns 3 as the CODELINE command appears on the 3rd line
DEBUG CODELINE()
```

CONSTANT

Type : Variable declaration command

Syntax : `CONSTANT <var> = <value>`

See Also : `GLOBAL`, `LOCAL`

Notes : None

Result : Define the variable **var** which contains the value held by **value**. This variable cannot be modified

Example :

```
// This example will set the variable A to 12 and prevents it from being modified
CONSTANT A%=12

// Subsequent uses with CONSTANT will be ignored
CONSTANT A%=13 // A% still contains 12

// Trying to modify the variable without CONSTANT will generate “error: assignment of read-only
// variable `__GLBASIC__::var” when compiling
A%=77
```

CONTINUE

Type : Loop processor

Syntax : CONTINUE

See Also : FOR, WHILE, REPEAT, NEXT, WEND, UNTIL

Result : Used with FOR, WHILE and REPEAT, this terminates the current iteration of the loop and starts the next one.

Example :

```
FOR X=1 TO 1000
    IF X=10
      CONTINUE // Reset a new loop when X = 10
    ENDIF

    ....
NEXT
```

COPYFILE

Type : File processor

Syntax : COPYFILE <source$>, <dest$>

See Also : None

Notes :

- Files cannot be copied out of or into a shoebox file
- If the destination file already exists, it is overwritten

Result : Copies the file held in **source$** to the file held in **dest$**. Nothing happens to the original file after copying

Example :

```
// This copies the file EXAMPLE.TXT to one called NEWEX.TXT
COPYFILE "EXAMPLE.TXT", "NEWEX.TXT"
```

COS

Type : Mathematical function

Syntax : `result = COS(<value>)`

See Also : ASIN, ACOS, ATAN, SIN, TAN

Notes : None

Result : Returns the cosine of an angle measurement, which is defined by **value**.

Example :

```
// Sets A to be 0.9998476952
A = COS(1.0)
```

CREATEDIR

Type : Directory creation

Syntax : `result% = CREATEDIR(<name$>)`

See Also : None

Notes :

- Linux and Mac can't create a directory if "~/" is used in **name$**. This short way of describing the user's home directory can only be used in the Terminal. If you need to write to the users home directory, then you need to convert this to a full path. If you need to use "~/" then you will need to use SHELLCMD

Result : Create a directory defined by **name$**. If the directory was created, this command returns **TRUE**.

If the directory could not be created, **FALSE** is returned

Example :

```
// This example creates the directory "TEST"
result = CREATEDIR("TEST")
```

CREATESCREEN

Type : Create virtual screen

Syntax : CREATESCREEN <index%>, <sprite%>, <width%>, <height%>

See Also : USESCREEN

Notes :

- The virtual screen must be selected using USESCREEN before it can be drawn on
- To delete a screen, use a width and height of 0
- On the Palm Pre 2, using the same screen with different sprites and sizes corrupts graphics

Result : Creates a virtual screen of the given size in pixels, with **width%** defining the horizontal size and **height%** the vertical size.

All drawing commands sent to this virtual screen will be performed on the sprite defined by **sprite%**

This sprite will also take the same width and height as the virtual screen.

The virtual screen number is defined by **index%**, and should be a value between 0 and 31.

Example :

```
// This example creates a virtual screen using index 0, and held in sprite 0. It is 32 pixels wide
// and 32 pixels high
CREATESCREEN 0, 0, 32, 32
```

DATA

Type : Store data within the program

Syntax : DATA <data1> [,<*data2*>,..<*dataN*>]

See Also : STARTDATA, ENDDATA, RESTORE, READ

Notes :

- Each data block MUST be started with STARTDATA and finished with ENDDATA, otherwise a "Syntax Error " compiler message will be generated during compilation
- All data within the data block MUST be a constant (no variables or calculations are allowed)

Result : Define a block of data (which can be a mixture of numbers and text), each of which must be separated by a comma.

This data can then be read using the READ command.

A specific set of data statements can be prepared for reading by using the RESTORE command

Example :

```
// An example of general usage of the DATA statement
DATA 1,2,3,"FOUR"

// To use data, it must be surrounded with STARTDATA and ENDDATA
STARTDATA info:
   DATA 1,2,3,"FOUR",5,6
ENDDATA
```

DEBUG

Type : Debugging command

Syntax : DEBUG <text$>

See Also : STDOUT, STDERR

Notes :
- This command only works in debug mode, and is completely ignored in release mode
- To send a new line code add '\n' to the text
- On some platforms (like the iPhone), STDOUT/STDERR should be used instead

Result : Writes the contents of text$ to the debug (output) window (which is the bottom part of the editor).

Example :

```
// This example sends some text to the debug window
DEBUG "THIS IS SOME TEXT"

// This example writes some text onto two lines
DEBUG "THIS IS LINE ONE\nTHIS IS LINE TWO"
```

DEC

Type : Decrease a value

Syntax : DEC <variable> [,*<amount>* | *<amount$>*]

See Also : INC

Notes :
- If a value for amount is not given, then 1 is used as default

Result : Decreases the value held in **variable**, by the value held in **amount**. If a string variable is used then **amount$** should be given, otherwise **variable$** will hold "-1".
Despite its description, DEC used with strings adds characters (and is equivalent to **variable$** = **variable$** + **append$**)

Example :

```
// This decreases A by 2
DEC A, 2
```

DECLARE

Type : External function declaration

Syntax : DECLARE(<functionName>, <dllName$>, (<paramPassed>), <return>)

See Also : DECLARE_ALIAS, DECLARE_C

Notes :

- DLL's are only used with Windows machines. It should be possible to use this command to load OSX's dylib and Linux's .o files
- This command assumes the function has a __stdcall calling convention

Result : Automatically load a DLL file when the program runs, and allows a function to be defined (with a list of appropriate parameters) so that it can be called later on.

functionName is the name of the function that will be called.

dllName$ is the name of the DLL file

paramPassed is a list of parameters (between two brackets) that define what parameters the function takes

return is the type of value the function returns

Example :

```
// This example loads the Display.DLL file, and defines it as requiring two integers (int's) and
// doesn't return anything.  The function name that can  be called is called 'displayName'
DECLARE(displayName, "Display.DLL", (int, int), void)
```

DECLARE_ALIAS

Type : External function declaration

Syntax : DECLARE_ALIAS(<functionName>, <dllName$>, <realName$>, (<paramPassed>), return)

See Also : DECLARE, DECLARE_C

Notes :

- DLL's are only used with Windows machines. It should be possible to use this command to load OSX's dylib and Linux's .o files
- This command assumes this function has a __stdcall calling convention

Result : Automatically load a DLL file when the program runs, and allows a function to be defined (with a list of appropriate parameters) so that it can be called later on. The real name of the function can be called by **functionName**.

functionName is the name of the function that will be called.

dllName$ is the name of the DLL file

realName$ is the exported filename of the DLL - usually (for C++) is a mangled name

paramPassed is a list of parameters (between two brackets) that define what parameters the function takes

return is the type of value the function returns

Example :

```
// This example loads the Display.DLL file, and defines it as requiring two integers (int's) and
// doesn't return anything.  Instead of using the mangled name, the function inside the DLL can be
// called using 'displayName'
DECLARE_ALIAS(displayName, "Display.DLL", "displayName%1234_absdd",(int,int),void)
```

DECLARE_C

Type : External function declaration

Syntax : DECLARE_C(<functionName>, <dllName$>, (<paramPassed>), <return>)

See Also : DECLARE_C_ALIAS, DECLARE

Notes :

- DLL's are only used with Windows machines. It should be possible to use this command to load OSX's dylib and Linux's .o files
- This command assumes this function has a __cdecl system

Result : Automatically load a DLL file when the program runs, and allows a function to be defined (with a list of appropriate parameters) so that it can be called later on.

functionName is the name of the function that will be called.

dllName$ is the name of the DLL file

paramPassed is a list of parameters (between two brackets) that define what parameters the function takes

return is the type of value the function returns

Example :

```
// This example loads the Display.DLL file, and defines it as requiring two integers (int's) and
// doesn't return anything.  The function name that can
// be called is called 'displayName'
DECLARE_C(displayName, "Display.DLL", (int, int), void)
```

DECLARE_C_ALIAS

Type : External function declaration

Syntax : DECLARE_C_ALIAS(<functionName>, <dllName$>, <realName$>, (<paramPassed>), <return>)

See Also : DECLARE_C, DECLARE

Notes :

- DLL's are only used with Windows machines. It should be possible to use this command to load OSX's dylib and Linux's .o files
- This command assumes this function has a __cdecl calling convention

Result : Automatically load a DLL file when the program runs, and allows a function to be defined (with a list of appropriate parameters) so that it can be called later on. The real name of the function can be called by **functionName**.

functionName is the name of the function that will be called.

dllName$ is the name of the DLL file

realName$ is the exported filename of the DLL - usually (for C++) is a mangled name

paramPassed is a list of parameters (between two brackets) that define what parameters the function takes

return is the type of value the function returns

Example :

```
// This example loads the Display.DLL file, and defines it as requiring two integers (int's) and
// doesn't return anything.  Instead of using the mangled
// name, the function inside the DLL can be called using 'displayName'
DECLARE_C_ALIAS(displayName, "Display.DLL", "displayName%1234_absdd",(int,int),void)
```

DECRYPT$

Type : Text decryption

Syntax : `text$ = DECRYPT$(code$, hex$)`

See Also : ENCRYPT$

Notes : None

Result : Returns the decrypted value of **hex$**. The parameter **code$** is used as a key.

Example :

```
LOCAL hex$, txt$
hex$ = ENCRYPT$("secr1t", "honey@moon.com")
txt$ = DECRYPT$("secr1t", hex$)

//hex$ holds =
"105d7bddc55ad98070922e2a05226c5266614425801ff379ea2a5c92ae9f686eefc896bbb88d19e0d3d36218b57ec2ef43c59d
8eb78c5780c0b95bfd16d34c0d7a7a06  5002b6c5c1"

PRINT "HEX:"+hex$+":\n\n",10, 10
PRINT "TXT:"+txt$+":\n", 10, 30
SHOWSCREEN
KEYWAIT
```

DEFAULT

Type : Option selector

Syntax : DEFAULT

See Also : SELECT, ENDSELECT

Notes : None

Result : If all previous CASE statements have not evaluated to **TRUE**, then all commands following DEFAULT will be executed

Example :

```
// Standard command layout
DEFAULT
    // Commands go here

// Basic layout
SELECT A
  CASE 1
        // If A = 1 then all code here is executed
   CASE 2
        // If A = 2 then all code here is executed
   DEFAULT
        // If A is neither 1 nor 2 then all code here will be executed
ENDSELECT
```

DELETE

Type : Item deletion

Syntax : DELETE <item>

See Also : FOREACH, DIMDEL

Notes :

- Use primary in a FOREACH/NEXT loop, this command is equivalent to

```
DIMDEL array[], index
CONTINUE
```

- No command in the same block will be executed after this command

Result : Deletes the array item from the array defined by the FOREACH loop

Example :

```
// This is the general use for the command
DELETE ITEM

// This example shows how to use the command in a loop
FOREACH NUM IN A[]
  // This will delete each item in the A[] array
  DELETE NUM
NEXT
```

DIM

Type : Array declaration

Syntax : `DIM <var>[<size1>][[<size2>]...[<size4>]]`

See Also : `DIMDEL`

Notes :

- Array indices start at 0, and continue up to size - 1
- To use this command in a function (or local to the main file), you need to initialise an empty array before using `DIM`
- Trying to access an element that is outside the size of an array will, in debug mode, create a "Out of DIM array" error message. In release mode, anything may happen - you could crash the program or operating system.
- There can be up to 4 dimensions. Trying to use more will make the compiler generate a "DIM has too many dimensions" error message
- Will try to use any currently allocated memory. If it doesn't fit any old memory is deleted and new memory is allocated. GLBasic will allocate a few entries more than that given by the size of a single element in order to avoid reallocation later

Result : Creates an array variable, defined by **var**, which contains **size1** elements.

Multi-dimensional arrays are allowed and the size of each further dimension should be given in square brackets, represented by ***size2*** to ***size4***

Using `DIM var[0]` de-allocates the memory reserved for the array.

Example :

```
// This item allocates 8 elements in the 'VAR' variable.  Access to the array starts at 0
DIM VAR[8]
VAR[0] = 3.5

// This example shows how to use DIM in a function.  This example defines the array and then
// allocates memory for the required number of indices
FUNCTION dummy:
LOCAL VAR[]; DIM VAR[8]
ENDFUNCTION
```

DIMDATA

Type : Initialise array with data

Syntax : DIMDATA <var>[],<value1>,...<valueN>

See Also : DIM, DIMDEL, REDIM

Notes :

- Variables as well as constant values can be used
- In release mode, the number of elements is limited to 24394 items.

Result : Allocates enough memory to store all the values from value1 to **valueN** into the array variable **var**, and stores these values in it

Example :

```
// This example allocates 3 elements for the variable 'VAR' and puts three numbers into each index
DIMDATA VAR[],1,2,3

// This example uses a variable in a DIMDATA array
A%=12
DIMDATA TEST%[],A%
// You would usually use DIMDATA like this to prevent compiler warning messages :
LOCAL a%[];  DIMDATA a%[],12,2
```

DIMDEL

Type : Item deletion

Syntax : DIMDEL <array>[],<index%>

See Also : DIM, DELETE

Notes : None

Result : Deletes the item held at position **index** in the **array[]** array . All items will be moved up accordingly.

If **index%** is less than 0, then the position will be counted from the end of the array forwards.

Example :

```
// This example deletes the 4th element in the array variable
DIMDEL array[],4
```

DIMPUSH

Type : Array addition

Syntax : DIMPUSH <array>[], <item%>

See Also : DIM, DIMDEL

Notes : None

Result : Adds the information held in item to the end of the array defined by **array[]**

Example :

```
// This example adds the contents of item into the array variable
DIMPUSH array[], item
```

DLLCALL

Type : External function declaration

Syntax : `DLLCALL(<name$>,<functionName$>,(void **) &<foo>);`

See Also : DECLARE, DECLARE_C, DECLARE_ALIAS, DECLARE_C_ALIAS

Notes : None

Result : This command, only used with `INLINE`, will manually load a DLL, defined by **name$** and call the function defined by **functionName$**.

(void **) &foo defines how the function is called, and works in conjunction with a __stdcall (or possibly __cdecl) declaration

Example :

```
FUNCTION test%:
INLINE
       void (__stdcall* foo)(int, char); // Declaration
       DLLCALL("test.dll","yo",(void **) foo);
ENDINLINE
ENDFUNCTION
```

DOESDIREXIST

Type : Directory presence checking

Syntax : `result% = DOESDIREXIST(<dir$>)`

See Also : DOESFILEEXIST

Notes : None

Result : Determines whether the directory held in **dir$** exists or not. If it does, then the command returns **TRUE**, otherwise **FALSE** is returned

Example :

```
// A is set to TRUE is the directory called “OUTPUT” exists in the current directory
A = DOESFILEEXIST(“OUTPUT”)
```

DOESFILEEXIST

Type : File presence checking

Syntax : `result% = DOESFILEEXIST(<file$>)`

See Also : `DOESDIREXIST`

Notes : None

Result : Determines whether the file held in **file$** exists or not. If it does, then the command returns **TRUE**, otherwise **FALSE** is returned

Example :

```
// A is set to TRUE is the file INFO.TXT exists in the current directory
A = DOESFILEEXIST("INFO.TXT")
```

DRAWANIM

Type : Sprite animation drawing

Syntax : `DRAWANIM <index%>, <tile%>, <x%>, <y%>`

See Also : `DRAWSPRITE`

Notes :

- If the sprite defined by **index%** doesn't exist or the tile index value is out of the sprites animation range, then nothing is displayed

Result : Draws a sprite from sprite index **index%**, with a tile index of **tile**. Horizontal positioning is set by **x**, and vertical positioning is defined by **y**.

Example :

```
// This draws frame 0 from sprite index 0 to position 100,100
DRAWANIM 0, 0, 100, 100
```

DRAWLINE

Type : 2D line drawing command

Syntax : DRAWLINE <sx%>, <sy%>, <ex%>, <ey%>, <colour%>

See Also : DRAWRECT

Notes : None

Result : This command draws a line from the starting horizontal and vertical pixel positions (stored in **sx** and **sy**) to the ending pixel positions (stored in **ex** and **ey**), using the colour defined by **colour**

Example :

```
// This draws a red diagonal line from the top left corner of the screen
DRAWLINE 0, 0, 100, 100, RGB(255, 0, 0)
```

DRAWRECT

Type : 2D rectangle drawing command

Syntax : DRAWRECT <sx%>, <sy%>, <ex%>, <ey%>, <colour%>

See Also : DRAWLINE

Notes : None

Result : This command draws a solid rectangle from the starting horizontal and vertical pixel positions (stored in **sx** and **sy**) to the ending pixel positions (stored in **ex** and **ey**), using the colour defined by **colour**

Example :

```
// This draws a red rectangle from the top left corner of the screen.  It is 100 pixels wide and 100
// pixels high
DRAWRECT 0, 0, 100, 100, RGB(255, 0, 0)
```

DRAWSPRITE

Type : Sprite drawing command

Syntax : DRAWSPRITE <index%>, <x%>, <y%>

See Also : DRAWANIM, USESCREEN

Notes :

- If the sprite defined by index doesn't exist, then nothing is displayed

Result : Draws a sprite from sprite index defined by **index%**. Horizontal positioning is set by **x**, and vertical positioning is defined by **y**.

Example :

```
// This draws sprite 0 at position 100, 100
DRAWSPRITE 0, 100, 100
```

ELSE

Type : Alternative action for comparison statements

Syntax : `ELSE`

See Also : `IF`, `ELSEIF`, `ENDIF`

Notes : None

Result : If a proceeding `IF`, `ELSE` or `ELSEIF` statement evaluated **FALSE**, then the statements following `ELSE` will be executed

Example :

```
IF A = 1
   ....
ELSE
   // This part will only be executed if A does NOT equal 1
   DEBUG "This will now be executed"
ENDIF
```

ELSEIF

Type : ELSE and IF combined

Syntax : `ELSEIF <var>`

See Also : `IF`, `ELSE`, `ENDIF`

Notes : None

Result : Evaluates **var**, and if **TRUE** executes the code directly following it. It is an easier way of writing :

```
ELSE
   IF var
   ENDIF
```

Example :

```
IF VAR = 1
   ....
ELSEIF VAR > 5
   // Do this part if VAR is greater than 5
   DEBUG "This will now be executed"
ENDIF
```

ENCRYPT$

Type : String encryption

Syntax : `result$ = ENCRYPT$(<key$>, <string$>)`

See Also : DECRYPT$

Notes : None

Result : Encode a plain-text string (held by **string$**), using the key value held by **key$**

Example :

```
// This command encrypts the text “this is a test” using the key string of “example”
DEBUG ENCRYPT$(“example”,”this is a test”)

// The resulting encryption string is :
e677929e284e3db97860261b50266e0cda4b6ec7d6196f4137a9a38da7e7ce1c93a5fc70b3f0666f
afe9bab3bbc65b80c456aa92d68bd69dda4b6ec7d6196f4137a9a38da7e7ce1c
```

END

Type : Program termination

Syntax : END

See Also : None

Notes :

- If the GLB_ON_QUIT subroutine is present, then everything in the subroutine will be executed when END is reached

Result : Immediately ends the program. Any commands following the END statement will not be executed

Example :

```
// This example ends the running program
END
DEBUG “HERE” // This command is not executed
```

ENDDATA

Type : Data block statement terminator

Syntax : ENDDATA

See Also : STARTDATA, DATA

Notes : None

Result : End a data block

Example :

```
// Define a data block, giving it the name of “letters”
STARTDATA letters:
  DATA “A”,”B”,”C”
ENDDATA
```

ENDFUNCTION

Type : Function terminator

Syntax : ENDFUNCTION

See Also : FUNCTION

Notes : None

Result : Denotes the end of a function. Without this, functions could run into each other causing compiler problems

Example :

```
// This is the standard use for ENDFUNCTION
ENDFUNCTION
```

ENDIF

Type : IF terminator

Syntax : `ENDIF`

See Also : `IF`, `ELSE`, `ELSEIF`

Notes : None

Result : Ends the nearest `IF` block. Without this, `IF` statements could run into each other, causing compiler problems

Example :

```
IF a = 1
  ...

// Close the IF statement
ENDIF
```

ENDINLINE

Type : C/C++ block terminator

Syntax : `ENDINLINE`

See Also : `INLINE`

Notes : None

Result : Denotes the end of the ability to add C/C++ code and functions. Without this, the compiler may have trouble with the next GLBasic command

Example :

```
// This ends the INLINE section
ENDINLINE
```

See `INLINE` for further examples.

ENDOFFILE

Type : File end checking

Syntax : `result% = ENDOFFILE(<handle%>)`

See Also : OPENFILE, CLOSEFILE, FILEPOSITION, FILESEEK

Notes : None

Result : This command returns **TRUE** if the file pointer of **handle** is at the end of the file, and returns **FALSE** if it is not.

Example :

```
// A will equal TRUE if the end of the file has been reached
A = ENDOFFILE(1)
```

ENDPOLY

Type : Polygon section terminator

Syntax : ENDPOLY

See Also : STARTPOLY

Notes : None

Result : This command terminates a STARTPOLY section. Without this, STARTPOLY commands could run into each other causing compiler or graphic problems

Example :

```
// This example creates a poly-vector starting at 1,1 and with a width and height of 8 pixels. It
// has purple colour
STARTPOLY 500
  POLYVECT 1,1, 8, 8,RGB(255,0,255)
ENDPOLY
```

ENDSELECT

Type : SELECT terminator

Syntax : ENDSELECT

See Also : SELECT

Notes : None

Result : Terminates the nearest SELECT statement. Without this, SELECT statements could run into each other, causing compiler problems

Example :

```
SELECT
  // Terminate the SELECT statement
ENDSELECT
```

ENDSUB

Type : Terminate a subroutine

Syntax : ENDSUB

See Also : SUB

Notes : None

Result : Denotes the end of a subroutine. Without this, subroutines could run into each other causing compiler problems

Example :

```
SUB
  ...
  // Terminate the subroutine
ENDSUB
```

ENDTYPE

Type : TYPE terminator

Syntax : ENDTYPE

See Also : TYPE

Notes : None

Result : Denotes the end of a user-created structure. Without this, other user-defined structures could run into each other causing compiler problems

Example :

```
TYPE
  ...
  // Terminate the TYPE section
ENDTYPE
```

EXPORT

Type : Function exporter

Syntax : EXPORT <functionName>

See Also : DECLARE_C, DECLARE_C_ALIAS

Notes : None

Result : Makes the function, defined by **functionName** available for use after loading the DLL.

This allows the function to be used in either GLBasic as a DLL or in another language.

The resulting export is a standard C function

Example :

```
// This example makes the AddFloat function available for use either using
// DECLARE_C or in another language
EXPORT AddFloat

FUNCTION AddFloat#: var1#, var2#
   RETURN var1+var2
ENDFUNCTION
```

FALSE

Type : Pre-defined variable

Syntax : `<command> | <var> = FALSE`

See Also : `TRUE`

Notes :

- This holds a value of 0

Result : Used to give a variable a specific value or to determine whether something is invalid/incorrect etc.

Example :

```
a%=FALSE
```

FILEPOSITION

Type : Retrieve file position

Syntax : `result% = FILEPOSITION(<handle%>)`

See Also : `OPENFILE`, `CLOSEFILE`

Notes : None

Result : Returns the file position (in bytes) of the opened file, defined by **handle%**

Example :

```
// This will return the read position of file handle
A = FILEPOSITION(HANDLE)
```

FILEREQUEST$

Type : Display file selection window

Syntax : `result$ = FILEREQUEST$(<open%>, <filter$>)`

See Also : None

Notes :

- For non-Windows machines, this command calls INPUT$
- This command does not work on Windows 7 x64

Result : Opens either a Open File dialogue window (**open%** = **TRUE**) or a Save File (**open%** = **FALSE**) dialogue window, and allows the user to select a file.

The list of available files is defined by **filter$**, which is a list of filters separated by a "|" character. Each filter is grouped in two (and separated by a '|' character) - the first part is a user-defined description of the second part, which is the file extension

Example :

```
// This example creates an Open File dialogue window presenting only files with
// the extension .TXT or .INI
FILES$ = FILEREQUEST$(TRUE, "Text *.TXT|*.INI")
```

FILESEEK

Type : Retrieve file position

Syntax : `FILESEEK <channel%>, <bytes%>, <direction%>`

See Also : `FILEPOSITON`

Notes : None

Result : Changes the file position of the opened file defined by **channel%**, by the value held in **bytes%**. The value held by **direction%** determines how bytes is used.

Value of **direction%**	How **bytes%** is interpreted
-1	Value is an offset from the end of the file
0	Value is an offset from the start of the file
1	Value is an offset from the current file position

Example :

```
// This example moves the file position 1024 bytes from the end of the file
FILESEEK 1, 1024, -1
```

FINALLY

Type : `TRY` terminator

Syntax : `FINALLY`

See Also : `TRY, THROW`

Notes : None

Result : Closes the nearest `TRY` statement. This is needed to prevent compiler errors

Example :

```
// The general usage for FINALLY is:
TRY
   // Code to be checked
CATCH
  // Process any errors here
FINALLY
```

FINDPATH

Type : Heuristic path finding

Syntax : `found% = FINDPATH (<map#[]>, <solution#[]>, <heuristic#>, <startx%>, <starty%>, <endx%>, <endy%>)`

See Also : None

Notes : None

Result : `FINDPATH()` searches the array **map#[]** for a path from **startx%**, **starty%** to **endx%**, **endy%**. An A* algorithm is used that can take costs of using various paths into consideration. A higher value in a given map location will cost more and should be avoided. A value of 0 and smaller is considered impassable. `FINDPATH` searches the map in 4 directions only, it does not calculate diagonal paths.

With the **heuristic** value you can specify whether the shortest route (=0.0) or the route with the lowest costs (=1.0) is to be calculated. You can use values in between 0 and 1 to have a trade-off between the two.

Imagine trying to get from one side of a high mountain to the other. The shortest path is straight over the top but will take you a lot of effort (i.e. the "cost"), the lowest cost one is to take the nice flat ground around the mountain, longer in terms of distance travelled, but easier.

The return value from `FINDPATH()` is the length of the found path.

If a path was found, the array **solution#[]** contains a 2 dimensional array. The first index indicates the number of the step. The second has an x (=index 0) and an y (=index 1) value of the step's position.

Example :

```
// --------------------------------
// Project:
// Start: Tuesday, September 23, 2003
// IDE Version: 1.30923
// Change this value for more/less terrain
MAXX = 500
MAXY = MAXX/2
scalex = 640 / MAXX
scaley = 320 / MAXY
DIM map[MAXX][MAXY]
DIM solution[0]
FOR x=0 TO MAXX-1; FOR y=0 TO MAXY-1; map[x][y]=100; NEXT; NEXT
FOR i=0 TO MAXX*MAXY*4; map[RND(MAXX-1)][RND(MAXY-1)] = RND(25); NEXT
FOR i=0 TO MAXY*.5; map[i+1][i] = 0; map[i+MAXY/2][MAXY-i-1]=0; NEXT
good = FINDPATH(map[], solution[], .3,  0,0,  MAXX-1, MAXY-1)
// Draw the map
FOR x=0 TO MAXX-1; FOR y=0 TO MAXY-1; DRAWRECT x*scalex, y*scaley, scalex, scaley, RGB(255*(1-map[x]
[y]), 255*map[x][y], 0); NEXT; NEXT

IF good
  PRINT "Found a solution!", 0, 400
  x=0; y=0
  FOR i=0 TO BOUNDS(solution[], 0)-1
    xl = x
    yl = y
```

```
    x = solution[i][0]
    y = solution[i][1]
    DRAWLINE xl*scalex+scalex/2, yl*scaley+scaley/2, x*scalex+scalex/2, y*scaley+scalex/2, RGB(0, 0,
255)
   NEXT
ELSE
  PRINT "No solution found!?", 0, 400
ENDIF

SHOWSCREEN
MOUSEWAIT
END
```

FMOD

Type : Mathematical operator

Syntax : `result = FMOD(<value>,<div>)`

See Also : MOD

Notes : None

Result : Returns the remainder of the result of **value** divided by **div**

Example :

```
// This sets A to be 3
A = FMOD(13, 5)
```

FOR

Type : Start a conditional loop

Syntax : FOR <var> = <start> TO <end> [*STEP <amount>*]

See Also : NEXT, STEP

Notes :

- The **var** variable is locally defined

Result : Initialises the counter **var**, initially giving it the value held in **start**.

When the NEXT terminator command is found, **var** is increased (or decreased) by the value held in **amount** (if present).

If the STEP command is not present, **var** is increased by 1.

The loop statement continues until the **end** value is reached or exceeded

Example :

```
// This example increases A by 1 with each iteration of the loop
FOR A = 1 TO 5
NEXT

// This example does exactly the same thing
FOR A=1 TO 5 STEP 1
NEXT
```

FORCEFEEDBACK

Type : Joystick feedback

Syntax : `FORCEFEEDBACK <index%>, <duration>, <xmotor>, <ymotor>`

See Also : None

Notes :

- Will also work with joysticks that have a rumble feature. For these, **xmotor** and **ymotor** should both be 1
- New values for this command will take precedence over previous values
- Excessively long and/or powerful values can damage both the joystick and hands

Result : This command activates force feedback on a joystick. A compatible joystick has one or more motors.

index% is the joystick index number, and should be between 0 and `GETNUMJOYSTICKS()` -1
duration is the amount of time feedback is performed. This is in milliseconds

xmotor and **ymotor** takes a value between -1 and 1 that indicates the power of the feedback

Example :

```
// This command activates force feedback on joystick 0 for 1000 milliseconds,
// with a power of 0.5 and 0.5
FORCEFEEDBACK 0, 1000, 0.5, 0.5
```

FOREACH

Type : Start array loop processing

Syntax : `FOREACH <item> IN <var[]>`

See Also : `NEXT`

Notes :

- Can only be used on single dimensional arrays
- **item** is locally defined and won't be available outside the loop

Result : Iterates through all the array items held in **var[]**, storing a pointer to each in item for access to each field

Example :

```
// This example loops through the items in the VAR array
FOREACH item IN VAR[]
NEXT
```

FORMAT$

Type : Numeric formatting

Syntax : `result$ = FORMAT$(numLetters, numAfterDot, number)`

See Also : None

Notes : None

Result : Formats **number** and returns the result.

The format is at least **numLetters** long (with spaces appended to the start if needed), and exactly **numAfterDot** digits after the full stop - zero's are added if needed

Example :

```
// This displays '    12.7'
A$ = FORMAT$(8, 1, 12.67)
```

FUNCTION

Type : Define a user-defined subroutine

Syntax : FUNCTION <name$> | <name> | <name%> | <name AS TYPE> :
[*<param1>,...<paramN>*]

See Also : ENDFUNCTION, RETURN

Notes :

- By convention, if a function isn't going to return anything, it should have an % appended to the name

Result : Defines a user-defined subroutine, with the name held in **name$**, **name** or **name%**.

Passed values are optional, and are stored in **param1** to **paramN**, and are defined local to the function

Default values to **param1** to **paramN** can be given - this then allows the parameter to be optional - if a parameter is not passed, then the default value is used. A CONSTANT cannot be used to pass a default value

A type can be returned, if the function is not in an extended type

Example :

```
// This example creates a function that requires 2 parameters to be passed
// to it
FUNCTION TEST : VALUE1, VALUE2
ENDFUNCTION

// This example allows the function to have an optional parameter.
// If no value is passed, then the default value of 12 is stored in the
// variable VALUE
FUNCTION TEST : VALUE = 12
ENDFUNCTION

// This example defines a function that returns an integer
FUNCTION TEST%:
ENDFUNCTION

// This example defines a function that returns a floating point value
FUNCTION TEST:
ENDFUNCTION

// This example defines a function that returns a string
FUNCTION TEST$:
ENDFUNCTION
```

GENFILE

Type : Retrieve vacant file handle

Syntax : `result% = GENFILE()`

See Also : None

Notes :
- The number will remain free until a file handle is opened
- Currently `GENFILE` returns a value between 0 and 7

Result : Finds the first free file handle number available to be used.

If no handles could be found, then this command returns -1

Example :

```
// This finds the first free file handle
handle% = GENFILE()
```

GENFONT

Type : Retrieve vacant font handle

Syntax : `result% = GENFONT()`

See Also : None

Notes :
- The number will remain free until a font is loaded

Result : Finds the first free font handle number available to be used.

If no handles could be found, then this command returns -1

Example :

```
// This finds the first free font handle
handle% = GENFONT()
```

GENSOUND

Type : Retrieve vacant sound handle

Syntax : `result% = GENSOUND()`

See Also : None

Notes :
- The number will remain free until a sound is loaded
- From V9.033 the value returned starts at 4095 and decreases for each use

Result : Finds the first free sound handle number available to be used.

If no handles could be found, then this command returns -1

Example :

```
// This finds the first free sound handle
handle% = GENSOUND()
```

GENSPRITE

Type : Retrieve vacant sprite handle

Syntax : `result% = GENSPRITE()`

See Also : None

Notes :
- The number will remain free until a sprite is loaded

Result : Finds the first free sprite handle number available to be used.

If no handles could be found, then this command returns -1

Example :

```
// This finds the first free sprite handle
handle% = GENSPRITE()
```

GENX_OBJ

Type : Retrieve vacant 3D object handle

Syntax : `result% = GENX_OBJ()`

See Also : None

Notes :

- The number will remain free until a 3D object is loaded

Result : Finds the first free 3D object handle number available to be used.

If no handles could be found, then this command returns -1

Example :

```
// This finds the first free 3D object handle
handle% = GENX_OBJ()
```

GETCOMMANDLINE$

Type : Command-line processor

Syntax : `result$ = GETCOMMANDLINE$()`

See Also : None

Notes :

- The first parameter will contain the program name, and usually the path to the program. On a Windows machine, running the program from the command line will not return the path.

Result : Retrieves any values passed to the program, each of which is separated by a space.

Example :

```
// This will show what parameters have been passed to the program
DEBUG GETCOMMANDLINE$()
```

GETCURRENTDIR$

Type : Retrieve current directory

Syntax : `result$ = GETCURRENTDIR$()`

See Also : None

Notes : None

Result : Returns the directory that is currently in use

Example :

```
// This returns the current working directory
CURRENTDIR$ = GETCURRENTDIR$()
```

GETDIGIX

Type : Retrieve HAT value

Syntax : `result = GETDIGIX(<index>)`

See Also : `GETDIGIY`

Notes : None

Result : This command returns the X joystick HAT movement value of the joystick indicated by index.

Example :

```
// This example gets the horizontal HAT value for joystick 0
A = GETDIGIX(0)
```

GETDIGIY

Type : Retrieve HAT value

Syntax : `result = GETDIGIY(<index>)`

See Also : `GETDIGIX`

Notes : None

Result : This command returns the Y joystick HAT movement value of the joystick indicated by index.

Example :

```
// This example gets the vertical HAT value for joystick 0
A = GETDIGIY(0)
```

GETENV$

Type : Retrieve system environment variable

Syntax : `result$ = GETENV$(<variable$>)`

See Also : `PUTENV`

Notes : None

Result : This command returns the value of the environment variable held in **variable$**. There are many different variables, all usually platform dependant.

Example :

```
// On a Windows platform, this command will return the complete search path used when locating
// files
A$=GETENV$("HOME")
```

GETFILE

Type : Read data from a file

Syntax : `GETFILE <fileName$>, <line%>, <store$>`

See Also : `PUTFILE`

Notes : None

Result : Reads a line from the file defined by **fileName$**, and stores the data in **store$**. The line number to read from is defined by line, and can be a value between 0 and 255

Example :

```
// This example reads in the 4th line of the file INFO.TXT
GETFILE "INFO.TXT",4,result$
```

GETFILELIST

Type : Retrieve list of files in current directory

Syntax : `count% = GETFILELIST(<find$>, <list$[]>)`

See Also : None

Notes :

- No pathing can be used - if you want to use a different directory to the current one, you need to use `SETCURRENTDIR` and then move back to the original one when you have finished
- The directory modifiers '. 'and '..' will be present

Result : Gets a list of all files and directories in the current directory. The result is stored in **list$[]**, whilst the total number of files and directories are returned, by combining the two.

This result equals the number of directories * 65536 + number of files - the number of directories is stored in the top 16 bits and the number of files in the lower 16 bits

find$ is a wildcat string, and determines what files are searched for.

Example :

```
// This example searches for all files in the current directory and stores then in the array called
// list$
num = GETFILELIST("*.*",list$[])
```

GETFILESIZE

Type : Retrieve file size of a program

Syntax : `result% = GETFILESIZE(<fileName$>)`

See Also : None

Notes :

- This command will return a value of 0 if the file is in use, or it doesn't exist

Result : Returns the size of the program defined by **fileName$**. This value is in bytes

Example :

```
// This returns the size of the file "INPUT.TXT" in bytes
DEBUG GETFILESIZE("INPUT.TXT")
```

GETFONTSIZE

Type : Retrieves the font size of currently used font

Syntax : `GETFONTSIZE <width%>, <height%>`

See Also : `SETFONT`

Notes : None

Result : Retrieves the width and height of one character from the currently used font, and stores the values in **width%** and **height%**, respectively.

Example :

```
// This example gets the current font size and stores the width in FX% and the height in FY%
GETFONTSIZE FX%, FY%
```

GETJOYBUTTON

Type : Joystick button state

Syntax : `result% = GETJOYBUTTON(<joystick%>, <index%>)`

See Also : None

Notes : None

Result : This command checks to see whether the joystick button, specified by **index%** is pressed on the joystick specified by **joystick%**.

If the button is pressed, then a value of 2 to the power of **index%** is returned. If no button is pressed, then 0 is returned

Example :

```
// This example returns 1 if joystick button 0 is pressed on joystick 0
A = GETJOYBUTTON(0,0)
```

GETJOYNAME$

Type : Joystick name

Syntax : `result$ = GETJOYNAME$(<index%>)`

See Also : `GETNUMJOYSTICKS()`

Notes : None

Result : This command returns the name of the joystick specified by the index **value%**. This value should be between 0 and `GETNUMJOYSTICKS()` - 1

Example :

```
// This example returns the name for joystick index 0
A$ = GETJOYNAME$(0)
```

GETJOYRX

Type : Joystick X movement value

Syntax : `result = GETJOYRX(<index%>)`

See Also : `GETJOYRY, GETJOYRZ`

Notes : None

Result : This command returns the X joystick movement value of the joystick indicated by **index%**.

Example :

```
// This example gets the horizontal value for the right stick on joystick 0
A = GETJOYRX(0)
```

GETJOYRY

Type : Joystick Y movement value

Syntax : `result = GETJOYRY(<index%>)`

See Also : `GETJOYRX, GETJOYRZ`

Notes : None

Result : This command returns the Y joystick movement value of the joystick indicated by **index%**.

Example :

```
// This example gets the vertical value for the right stick on joystick 0
A = GETJOYRY(0)
```

GETJOYRZ

Type : Joystick Z movement value

Syntax : `result = GETJOYRZ(<index%>)`

See Also : GETJOYRX, GETJOYRZ

Notes : None

Result : This command returns the Z joystick movement value of the joystick indicated by **index%**.

Example :

```
// This example gets the Z value for the right stick on joystick 0.  Z is usually a slider of some
// sort
A = GETJOYRZ(0)
```

GETJOYX

Type : Joystick X movement value

Syntax : `result = GETJOYX(<index%>)`

See Also : GETJOYY, GETJOYZ

Notes : None

Result : This command returns the X joystick movement value of the joystick indicated by **index%**.

Example :

```
// This example gets the horizontal value of joystick 0
A = GETJOYX(0)
```

GETJOYY

Type : Joystick Y movement value

Syntax : `result = GETJOYY(<index%>)`

See Also : GETJOYX, GETJOYZ

Notes : None

Result : This command returns the Y joystick movement value of the joystick indicated by **index%**.

Example :

```
// This example gets the vertical value of joystick 0
A = GETJOYY(0)
```

GETJOYZ

Type : Joystick Z movement value

Syntax : `result = GETJOYZ(<index%>)`

See Also : GETJOYX, GETJOYY

Notes : None

Result : This command returns the Z joystick movement value of the joystick indicated by **index%**.

Example :

```
// This example gets the Z value (usually a slider) of joystick 0
A = GETJOYZ(0)
```

GETLASTERROR$

Type : Retrieve last error information

Syntax : `result$ | result% = GETLASTERROR$()`

See Also : CLEARERROR

Notes :

- The beta of V7 does not recognise this command and generates a “Syntax Error”
- In V8.148, this is returned if LOADSOUND/LOADSPRITE etc. is used with file that is not present

Result : Returns the error number and string of the last error that occurred. The possible error results are :

Possible values from GETLASTERROR$() :
0 success
3 file does not exist
4 wrong argument
5 string is too long
7 not enough memory
8 wrong dimension
9 out of dimensioned range (makes the program end)
20 no more data available

Example :

```
// This example puts the last error details into A$
A$ = GETLASTERROR$()
```

GETMOUSECOUNT

Type : Attached mouse information

Syntax : `result% = GETMOUSECOUNT()`

See Also : `SETACTIVEMOUSE`

Notes :

- Currently, this command is only valid for the iPhone/iPod/iPad and webOS, and all other operating systems will return an undefined value.

Result : Returns the number of attached mouse input devices

Example :

```
// This example gets the number of mice attached to the computer
num% = GETMOUSECOUNT()
```

GETNUMJOYSTICKS

Type : Get number of joysticks

Syntax : `result% = GETNUMJOYSTICKS()`

See Also : `GETJOYNAME$`

Notes :

- Adding or removing joysticks whilst the program is running will not change the value returned from this command

Result : This command returns the number of joysticks attached to the computer when the program starts.

Example :

```
// This command returns the number of joysticks attached to the computer
A% = GETNUMJOYSTICKS()
```

GETORIENTATION

Type : Get rendering orientation

Syntax : `result% = GETORIENTATION()`

See Also : `SETORIENTATION`

Notes : None

Result : This command returns orientation that all rendering is performed on.

result% can be one of the following :

Value of **result%**	Orientation :
0	Up (Portrait)
1	Right (Landscape)
2	Down (Inverse Portrait)
3	Left (Inverse Landscape)

Example :

```
// Put the current oritentation in the Output window
DEBUG GETORIENTATION()
```

GETPIXEL

Type : Read colour of a pixel

Syntax : `result% = GETPIXEL(<x>,<y>)`

See Also : LOADBMP, USEASBMP, SHOWSCREEN, SETPIXEL

Notes : None

Result : This command returns the pixel colour at the positions stored in **x** and **y**.

With 16-bit modes, the colour read may not correspond exactly to any colour written due to conversion between 32-bit colour and 16-colour modes.

It should also be noted that this command is very slow.

Example :

```
// This reads the pixel at coordinates 100 by 100
RESULT = GETPIXEL(100, 100)
```

GETSCREENSIZE

Type : Screen resolution information

Syntax : GETSCREENSIZE <width%>, <height%>

See Also : SETSCREEN

Notes : None

Result : This command retrieves the current screen pixel size, storing the width in **width%** and the height in **height%.**

The value read is from either the size set by the SETSCREEN command, or if that has not been used, by the project options.

If the program has been compiled as a console program, this command will try to find the visible number of columns (stored in **width%**) and rows (stored in **height%**) for the console window.

Example :

```
// This example gets the current screen size and stores them in WIDTH% and HEIGHT%
GETSCREENSIZE WIDTH%, HEIGHT%
```

GETSPRITESIZE

Type : Sprite size information

Syntax : `GETSPRITESIZE <index%>, <width%>, <height%>`

See Also : None

Notes :

- If the sprite defined by index doesn't exist, then **width%** and **height%** will contain 0

Result : Retrieves the sprite width and height. The sprite to be used is defined by **index%**. The sprite's width is stored in **width%**, and the height is stored in **height%**.

Example :

```
// This example stores the sprite width and height from index 1 and stores it in W% and H%
GETSPRITESIZE 1, W%, H%
```

GETTIMER

Type : Time since `SHOWSCREEN` was used

Syntax : `result = GETTIMER()`

See Also : `SHOWSCREEN`, `GETTIMERALL`

Notes :

- The value returned is a decimal number

Result : Returns (in 1/1000 of a second), the time since the `SHOWSCREEN` command was last called

Example :

```
// This example gets the current time since the last SHOWSCREEN command
A = GETTIMER()
```

GETTIMERALL

Type : Time since program started

Syntax : `result = GETTIMERALL()`

See Also : `GETTIMER`

Notes :

- The value returned is a floating-point value

Result : Returns (in 1/1000 of a second), the time since the program was first started

Example :

```
// This example gets the time since the program started
A = GETTIMERALL()
```

GETVIEWPORT

Type : Get viewport size

Syntax : `GETVIEWPORT <x%>, <y%>, <width%>, <height%>`

See Also : `VIEWPORT`

Notes : None

Result : This command returns the current viewport size. It stores the starting X and Y coordinates in **x %** and **y%**, and the width and height is stored in **width%** and **height%**.

If the viewport has been set to 0, 0, 0, 0, then the actual viewport size (the current screen width and height) is returned

Example :

```
// This example returns the current viewport size
GETVIEWPORT X%,Y%,WIDTH%,HEIGHT%
```

GLOBAL

Type : Define a global variable

Syntax : GLOBAL <variable [*[]*]> | <variable$ [*[]*]> | <variable% [*[]*]>

See Also : LOCAL

Notes :

- GLOBAL can currently be used in functions, but as program execution may not work correctly

Result : Defines **variable** as being global. This means that the variable can be accessed and modified from all functions and subroutines within the file they are created.

This means that GLOBAL variables are accessible only in the source file that they are created, except for those created in the Main Program file, which are accessible in all files.

Putting GLOBAL in front of a variable before accessing it, will force the global version of a variable to be used, even if a LOCAL version exists.

Any type of variable can be defined as GLOBAL.

Example :

```
// This example defines the variable ITEM as being global and accessible
// everywhere
GLOBAL ITEM

// A test function that uses the global version of the ITEM variable, and not the locally defined one
FUNCTION test:
LOCAL ITEM

DEBUG GLOBAL ITEM  // Output the value of the global variable, and not the local one
ENDFUNCTION
```

GOSUB

Type : Call a subroutine

Syntax : `GOSUB <name>`

See Also : `SUB, ENDSUB, GOTO`

Notes : None

Result : Calls the subroutine **name**, and when it has finished, continues program execution at the next statement after the GOSUB command

Example :

```
// This example calls the subroutine NAME
GOSUB NAME
// Execution continues here after the subroutine has finished
 ...

// Which is defined here, and run when called
SUB name:
ENDSUB
```

GOTO

Type : Jump to a label

Syntax : GOTO <name>

See Also : GOSUB

Notes : None

Result : This command causes program execution to jump to the defined label **name**, and continue execution from there, rather than continuing with the next command.

GOTO is depreciated. Uninhabited use will make your program hard to understand and debug. Its ability to control program follow can be handled better by other commands.

Example :

```
// This example jumps to the routine called NAME
GOTO NAME
// Nothing is executed here
....

// Which is defined here
NAME:
.... // Program execution continues here
```

GRABSPRITE

Type : Create a sprite

Syntax : `GRABSPRITE <index%>, <x%>, <y%>, <width%>, <height%>`

See Also : None

Notes :

- On an iPhone, this command is slower in Landscape mode than in Portrait mode
- Using `GRABSPRITE <index%>,0,0,0,0` frees allocated memory

Result : Grabs a portion of the back-buffer and creates a sprite defined by **index**, which the graphic is stored in.

The horizontal position is defined by **x**, and the vertical by **y**.

The width of the area to be grabbed is defined by **width**, and the height by **height**.

Example :

```
// This example grabs the contents of the back-buffer from 16, 16 and for 32 pixels along and 32
// pixels down and stores the  graphic in sprite index 2
GRABSPRITE 2, 16, 16, 32, 32
```

HIBERNATE

Type : Make program sleep

Syntax : HIBERNATE

See Also : SLEEP

Notes : None

Result : Makes the program sleep until the user performs an interactive action like pressing a key, moving a mouse etc.

Example :

```
// Makes the program sleep unless user performs some action
HIBERNATE
```

HUSH

Type : Stop all sounds

Syntax : HUSH

See Also : LOADSOUND, PLAYSOUND, ISMUSICPLAYING, STOPMUSIC

Notes : None

Result : Any sounds being played (using PLAYSOUND) will be stopped

Example :

```
// This example stops all sounds currently being played
HUSH
```

IF

Type : Evaluate condition

Syntax : `IF <evaluation> ; result; ENDIF | IF <evaluation> THEN result`

See Also : `THEN`, `ELSE`, `ELSEIF`, `ENDIF`

Notes : None

Result : Evaluates **evaluation**, and if it is **TRUE**, the code (result) following the `IF` statement is executed. If no `ELSE`/`ELSEIF` statements are present, then no further tests are performed

Example :

```
// This example shows a multi-line IF/ENDIF setup
IF A = 1
  DEBUG “A is 1”
ENDIF

// This example shows a single-line IF/THEN setup
IF A=1 THEN DEBUG “A is 1”
```

IMPORT

Type : Make external function available

Syntax : `IMPORT *["C"]* __stdcall type function(*[param1],[..paramN]*);`

See Also : `REQUIRE`

Notes :

- When using a pointer, the '*' symbol needs to be next to the pointer type, and not the variable (or function name), i.e. :
```
IMPORT "C" const char* tmpnam(const char*) and NOT
IMPORT "C" const char *tmpnam(const char *)
```

Result : Makes available both functions compiled in from using REQUIRE or any system calls. "C" and __stdcall are optional, but are usually needed for using C functions. If using a C++ function, "C" is usually not needed.

type is the standard C/C++ type definition, which is one of the following : int, char, float, double function is the name of the function that is to be used, and may take one or more parameters (**param1** to **paramN**)

Example :

```
// This example allows access to the test function which takes no parameters and returns nothing
// either
IMPORT "C" __stdcall void test(void);

// This example allows access to Windows MessageBox API
IMPORT "C" __stdcall MessageBoxA(int hwnd, const char* text, char* caption, int flags);
```

IN

Type : Array loop processor

Syntax : IN <array[]>

See Also : FOREACH, NEXT

Notes : None

Result : Defines the FOREACH command to use the array defined by **array[]**

Example :

```
// The basic usage is as follows :
IN array[]

// The following example shows defines the LOOP array as being the list to use
FOREACH ITEM IN LOOP[]
  ...
NEXT
```

INC

Type : Increase a value

Syntax : INC <variable | variable$> [, *amount* | *amount$*]

See Also : DEC

Notes :

- If a value for amount is not given, then 1 is used as default

Result : Increases the value held in **variable** (or **variable$**), by the value held in **amount** or **append$**

If a string is used, then **append$** is appended to it. This is another way of writing **variable$** = **variable$** + **append$**. With strings, **append$** should be given, otherwise "1" will be put into **variable$**

Example :

```
// This increases A by 1
INC A, 1

// This appends "Hello" to A$
INC A$,"Hello"
```

INIGET$

Type : Retrieve data from an initialisation (.INI) file

Syntax : `result$ = INIGET$(<section$>, <key$> [, <noData$>])`

See Also : INIOPEN, INIPUT

Notes :

- Values held in section$ and key$ are not case sensitive

Result : Retrieves data stored in the currently opened INI file.

The data retrieved is from the section defined by **section$**, and the key name held by **key$.**

If neither **section$** or **key$** exist, then the value held by **noData$** is returned. If **noData$** isn't present, then the string "NO_DATA" is returned

Example :

```
// This example returns the value from section "KEY" and key value of "VALUE"
STORE$ = INIGET$("KEY","VALUE")
```

INIOPEN

Type : Open an initialisation file (.INI) for reading and writing

Syntax : `INIOPEN <filename$>`

See Also : `INIGET$`, `INIPUT`

Notes :

- If the file held by **fileName$** doesn't exist, then it will be created on most platforms. However, this wont happen on Palm and Android platforms unless an empty file is create first.
- An example INI file :

```
[DISPLAY]
SCREENWIDTH=1280
SCREENHEIGHT=768
SCREENFULL=0
```

  DISPLAY is the setion name and **SCREENWIDTH**, **SCREENHEIGHT** and **SCREENFULL** are the key values

Result : Opens the initialisation file stored in **fileName$**. If a previous INI file has already been opened, then its contents will be written to disk before the new file is opened.

If **fileName$** is an empty string, any modifications using `INIPUT` will be written to disk.

Example :

```
// This example opens the INI file called INFO.INI
INIOPEN "INFO.INI"
```

INIPUT

Type : Write to an initialisation file (.INI)

Syntax : `INIPUT <section$>, <key$>, <data$>`

See Also : `INIOPEN`, `INIGET$`

Notes : None

Result : Write the data held in data$, in the specified section held by **section$**, and with the specified key, held by **key$**

Example :

```
// This example writes VALUE to the section called SECTION, and a key name of KEY
INIPUT "SECTION","KEY","VALUE"
```

INKEY$

Type : Retrieve a key

Syntax : `result$ = INKEY$()`

See Also : `KEY`

Notes :

- If no key is pressed, then this command returns an empty string
- Can only be used with platforms with a keyboard – otherwise an empty string is returned

Result : Returns the keyboard character pressed by the user

Example :

```
// This example gets the current key pressed by the user
A$ = INKEY$()
```

INLINE

Type : Start C/C++ processing

Syntax : `INLINE`

See Also : `ENDINLINE`

Notes :

- `INLINE` should not be placed in the main program file, but in a source program file

Result : Initialises the ability to add C/C++ commands.

Example :

```
// This is the layout of  INLINE/ENDINLINE
INLINE
  ...
ENDINLINE

// This example creates  GLBasic function but uses C to return a result
FUNCTION cTest%:
INLINE
   return 1;
ENDINLINE
ENDFUNCTION

// This example includes a Windows header
INLINE
   extern "C" __stdcall int GetSystemMetrics(int);
ENDINLINE
```

INPUT

Type : Wait for input

Syntax : `INPUT$ <result$>, <x%>, <y%>`

See Also : `KEYWAIT`, `INKEY$`

Notes :

- Currently on a Windows CE machine, the input caret is incorrectly placed

Result : Interrupts program execution to display a flashing cursor are the co-ordinates specified by **x** and **y**.

The user can then use the keyboard to enter text until the RETURN key is pressed, and which point all entered text is stored in **result$**.

Program execution will then continue.

Example :

```
// This example waits for the user to input some text
INPUT A$, 100, 100
```

INSTR

Type : Find characters in a string

Syntax : result% = INSTR(<word$>, <find$> *[, <pos%>]*)

See Also : ASC, CHR$, DECRYPT$, ENCRYPT$, FORMAT$, LCASE$, LEFT$, LEN, LTRIM$ MID$, REPLACE$, RIGHT$, RTRIM$, SPLITSTR, TRIM$, UCASE$

Notes : None

Result : Searches **word$** to find the characters held in **find$**.

If at least one occurrence is found, its position within **word$** is returned.

If no match has been found, then -1 is returned instead

The option parameter of **pos%** defines the initial index position of where the search takes place. A value of 0 starts the search at the first character position.

Example :

```
// This returns the first position of X, which is at position 1 (the letter E
// is at position 0)
LOCAL Position
Position = INSTR("EXAMPLE", "X", 0)
PRINT Position,10,10
SHOWSCREEN
KEYWAIT
OUTPUT: 1

// Find the position of X in a string (No Start Position used)
// This returns the first position of X, which is at position 1 (the letter E
// is at position 0) here no start position value is used so the position starts
// from 0
LOCAL Position
Position = INSTR("EXAMPLE", "X")
PRINT Position,10,10
SHOWSCREEN
KEYWAIT
OUTPUT: 1

// Find the position of AMP in a string.
// This returns the first position of AMP, which is at position 2 (the letter E
// is at position 0)
LOCAL Position
Position = INSTR("EXAMPLE", "AMP", 0)
PRINT Position,10,10
SHOWSCREEN
KEYWAIT
OUTPUT: 2
```

INTEGER

Type : Integer conversion

Syntax : `result% = INTEGER(<value>)`

See Also : None

Notes : None

Result : This command truncates a real number (**value**) to the lowest integer value

Example :

```
// This sets A to be 13
A = INTEGER(13.45)
```

ISFULLSCREEN

Type : Status of graphics screen

Syntax : `result% = ISFULLSCREEN()`

See Also : SETSCREEN

Notes :

- This command currently returns **FALSE** on webOS machines

Result : This command returns **TRUE** if the graphics screen is running in full screen mode, or **FALSE** if the screen is running in a window

Example :

```
// This example returns TRUE if the program is using a full screen
RESULT% = ISFULLSCREEN()
```

ISMUSICPLAYING

Type : Return status of music

Syntax : `result% = ISMUSICPLAYING()`

See Also : LOADSOUND, PLAYSOUND

Notes : None

Result : Returns **TRUE** if music is playing, or **FALSE** if it is not.

Example :

```
// This example returns TRUE if music is currently being played
A = ISMUSICPLAYING()
```

JOYSTATE

Type : Set joystick state

Syntax : `JOYSTATE <x>, <y>, <b1>, <b2>`

See Also : None

Notes : None

Result : This command sets the current joystick to specified values. X and Y values are specified by **x** and **y**, whilst **b1** represents button A and **b2** represents button B.

If **b1** and/or **b2** = **TRUE**, then a button is pressed.

The X and Y values can be :

Value of **x** or **y**	Direction
-1	Left or up
0	No direction
1	Right or down

Example :

```
// This example sets the joystick values for joystick 0, to position 100,100 and with no  buttons
// pressed
JOYSTATE 100, 100, FALSE, FALSE
```

KEY

Type : Returns status of key pressed

Syntax : `result% = KEY(<code%>)`

See Also : `KEYWAIT`, `INKEY$`, `INPUT$`

Notes :

- On machines with no keyboard, this command will always return **FALSE**
- This command should not be used when compiling a console program. `INKEY`, `KEYWAIT` or `INPUT` should be used instead

Result : Returns the status of the key pressed, as defined by the scan-code code.

If the key is pressed, then result will equal **TRUE**, otherwise it will equal **FALSE**.

The scan code is platform specific - see this list for some codes

Example :

```
// This example returns TRUE is the user is pressing the +/= key (on a PC)
A = KEY(13)
```

KEYWAIT

Type : Wait for a key press

Syntax : `KEYWAIT`

See Also : `MOUSEWAIT`

Notes :

- There is no way of detecting what key was pressed

Result : Any program is interrupted until the user presses a key on the keyboard

Example :

```
// This example pauses the program until the user presses a key
KEYWAIT
```

KILLFILE

Type : Delete a file

Syntax : `KILLFILE <fileName$>`

See Also : None

Notes : None

Result : Deletes the file stored in **fileName$**

Example :

```
// This example deletes the file called INFO.TXT
KILLFILE "INFO.TXT"
```

LCASE$

Type : Convert character to lower-case

Syntax : result$ = LCASE$(<text$>)

See Also : UCASE$

Notes : None

Result : Converts all upper case characters (A to Z) held in **text$**, into their lower case equivalent (a to z). All other characters are left as they are.

Example :

```
// This example converts the text to lower case letters and stores it in A$
// The variable would be holding 'this is a test'
LOCAL A$
A$ = LCASE$("THIS IS A TEST")
PRINT A$, 10, 10
SHOWSCREEN
KEYWAIT
```

OUTPUT: this is a test

LEFT$

Type : Get number of characters from left-hand side of a string

Syntax : `result$ = LEFT$(<text$>, <num%>)`

See Also : `RIGHT$`, `MID$`

Notes : None

Result : This command returns a given number of characters, defined by **num%**, from the left-hand side of the string defined by **text$**

If **num%** = 0, then an empty string is returned

If **num%** exceeds the number of characters in **text$**, then the complete contents of **text$** is returned

Example :

```
// This example would display "THIS"
LOCAL A$
A$ = LEFT$("THIS IS A TEST",4)
PRINT A$, 10, 10
SHOWSCREEN
KEYWAIT
```

OUTPUT: THIS

LEN

Type : Get number of characters in a string or the text width of a string

Syntax : `result% = LEN(<text$> [, <bKernal%> = FALSE])`

See Also : None

Notes :

- An ASCII value of 0 is counted as a character, except when **bKernal%** = **FALSE**

Result : This command returns the length (number of characters) in **text$** if a value for **bKernal%** is not given. If **bKernal%** is set to **TRUE**, then the value returned is the proportional text width. If **bKernal%** is set to **FALSE**, then the value returned is the non-proportional text width.

If **text$** is an empty string, then the value returned is 0 in all cases

Example :

```
// This sets A to be 7, which is the number of characters in the word 'EXAMPLE'
LOCAL A
A = LEN("Example")
PRINT A, 10, 10
SHOWSCREEN
KEYWAIT
```

OUTPUT: 7

```
// This example returns the length of "Example" using the current proportional font
DEBUG LEN("Example,TRUE")
```

OUTPUT : 82

Note that this value may change depending on font used.

LET

Type : Allow assignment of a variable

Syntax : `LET <var> = value`

See Also : None

Notes :
- Command is obsolete

Result : Assigns the value held in **value** to the variable **var**.

Example :

```
// Set A to equal 5
LET A = 5

// A slightly quicker way is :
A=5
```

LIMITFPS

Type : Set running program's speed

Syntax : `LIMITFPS <fps>`

See Also : None

Notes :
- A fps value of -1 uses a frame rate of 500

Result : This command limits the running program to the frames per second value stored in **fps**

Example :

```
// This example limits the frames per second to 500
LIMITFPS -1
```

LOADANIM

Type : Load a graphic file to split into cells

Syntax : LOADANIM <fileName$>, <index%>, <width%>, <height%>

See Also : DRAWANIM, LOADSPRITE, LOADSPRITEMEM

Notes :

- Giving an empty filename (or a filename that does not exist) will delete the graphic from memory
- PNG, BMP and JPG files are supported
- An animated sprite should only be used be displayed using DRAWANIM, as using DRAWSPRITE may not

Result : Loads the graphic file held in **fileName$**, and stores it in the sprite index stored in **index%**.

The graphic is internally split into **width%** pixels horizontally and **height%** pixels vertically, for use with the DRAWANIM command

Example :

```
// This loads the animated graphic file into index 1 and splits it into columns and rows of 5 pixels
LOADANIM "PLAYERANIIM.BMP", 1, 5, 5
```

LOADBMP

Type : Load a graphic file and use it as a background

Syntax : LOADBMP <fileName$>

See Also : None

Notes :

- Currently only BMP and PNG files are supported

Result : Loads the graphic file called **fileName$**, which will then be used as a background for each call to SHOWSCREEN

Example :

```
// This example loads the BACKGROUND.BMP and uses it as the background
LOADBMP "BACKGROUND.BMP"
```

LOADBUMPTEXTURE

Type : Load a grey scale height map

Syntax : LOADBUMPTEXTURE <fileName$>, <index%>

See Also : X_SETTEXTURE

Notes :

- Giving an empty filename (or a filename that does not exist) will delete the graphic from memory
- PNG, BMP and JPG files are supported

Result : Loads the graphic file called **fileName$**, which is treated as a height-map. A normal map is then created for it.

The loaded graphic is stored in the graphic index defined by **index%**.

Bump textures are used to make 3D objects more realistic, by simulating bricks, for example.

Example :

```
// This example loads the bump texture into index 1
LOADBUMPTEXTURE “BUMP.BMP”, 1
```

LOADFONT

Type : Load a font file

Syntax : `LOADFONT <fileName$>, <index%>`

See Also : `SETFONT`

Notes :

- Giving an empty filename (or a filename that does not exist) will delete the font from memory
- Font width must be a multiple of 16 pixels and height must be multiples of 8 bytes
- Characters can be proportional or fixed width with GLBasic V8.148 onwards

Result : Loads the graphic file called **fileName$**, into the font index **index%**. The font index then allows the font to use used to display characters.

The value of **index%** should be between 0 and 30

Example :

```
// This example loads the font into font index 1
LOADFONT "LARGEFONT.PNG", 1
```

LOADSOUND

Type : Loads a WAV file (PCM format only)

Syntax : LOADSOUND <fileName$>, <index%>, <buffer%>

See Also : PLAYSOUND

Notes :

- Giving an empty filename (or a filename that does not exist) will delete the sound from memory
- An invalid WAV file will be loaded, but wont play

Result : Loads the WAV file called **fileName$**, into the sound index **index%**. The command then allows the sound to be played.

The value held by **buffer%** can be between 1 and 4 and specified the number of times a sound can be played without being cut off. The higher the value, the more memory used to perform the action

Example :

```
// This example loads the WAV file into the sound index 1
LOADSOUND "WAIL.WAV", 1, 1
```

LOADSPRITE

Type : Load a graphic file

Syntax : `LOADSPRITE <fileName$>, <index%>`

See Also : `DRAWSPRITE, LOADANIM`

Notes :

- Giving an empty filename (or a filename that does not exist) will delete the graphic from memory
- PNG, BMP and JPG files are supported

Result : Loads the graphic file called **fileName$**, into the sprite index **index%**. The command then allows the sprite to be displayed, using `DRAWSPRITE`

Example :

```
// This example loads the file into the sprite index 1
LOADSPRITE "PLAYER.PNG", 1
```

LOADSPRITEMEM()

Type : Loads a graphic file into an array

Syntax : `result% = LOADSPRITEMEM(<fileName$>, <width%>, <height%>, <array%[]>)`

See Also : DRAWSPRITE, LOADANIM, LOADSPRITE, MEM2SPRITE

Notes :

- The format of the data held in the array is red, green, blue and alpha

Result : Loads the graphic file called **fileName$**, into the array defined by **array%[]**. The width and height of the sprite to be loaded is defined by **width%** and **height%**

If the command completed successfully, then **TRUE** is returned. If there was a problem, **FALSE** will be returned instead

Example :

```
// This example loads the sprite called “test.png”, which is 128 x 128 pixels into the array called
// data
result%=LOADSPRITEMEM(“test.png”,128,128,data%[])
```

LOCAL

Type : Define a local variable

Syntax : LOCAL <variable>

See Also : GLOBAL

Notes : None

Result : Defines **variable** as being local. This means that the variable can only be accessed and modified from either the main program (if it was defined there) or a SUB or function (if it was defined there).

Memory allocated by LOCAL will be released when the main section of the code it was created in, finishes.

Using LOCAL in front of a variable when accessing it will force the use of the local version of the variable. If this command isn't present then the local version will be used if it exists, but the GLOBAL version will be used if the LOCAL version doesn't exist.

Example :

```
LOCAL item

// This example defines the local variable 'ping' which can only be used inside the function 'test'
FUNCTION test:
LOCAL PING
ENDFUNCTION
```

LOGN

Type : Compute natural logarithm

Syntax : `result = LOGN <value>`

See Also : `POW`

Notes :

- To calculate a logarithm to another base use :

```
log_to_base_b_of_x = LOGN(x) / LOGN(b)
```

- In order to calculate the inverse value (anti-log) you have to calculate `EXP(1)`, where the `EXP` function can be expressed as `POW(2.718281828, 1)`
- The natural number ("E") is approximately 2.718281828

Result : Calculates the natural logarithm to the base "E", of **value**

Example :

```
DEBUG LOGN(1)         // This produces a value of 0
DEBUG LOGN(2.5)       // This produces a value of 0.9162907319
DEBUG LOGN(0)         // This produces -1.#INF
```

LOOPMOVIE

Type : Continually loop a movie

Syntax : `LOOPMOVIE <fileName$>`

See Also : `PLAYMOVIE`

Notes :

- This command currently only works on Windows platforms

Result : Loads and plays the movie file **fileName$**, and continually replays it until the SPACEBAR is pressed.

Example :

```
// This loops the movie 'MOVIE.AVI'
LOOPMOVIE "MOVIE.AVI"
```

LTRIM$

Type : Trims text from the beginning of text

Syntax : `result$ = LTRIM$(<text$> [, <remove$="\t\r\n\v\f">])`

See Also : `TRIM$`, `RTRIM$`

Notes : None

Result : This removes all text held in **remove$** from the text in **text$**.

Text is removed from the left-hand side of the text only.

Example :

```
// This example removes all "a"'s from the start of the text "abcdefa"
LOCAL Trimmed$
Trimmed$ = LTRIM$("aaaabacdefa","a")
PRINT Trimmed$, 10, 10
SHOWSCREEN
KEYWAIT
```

OUTPUT: bacdefa

MAX

Type : Returns the larger of two values

Syntax : `result = MAX(<value1>,<value2>)`

See Also : `MIN`

Notes : None

Result : Returns the larger value of either **value1** or **value2**

Example :

```
// This sets A to be the highest of the two values, which is 17
A = MAX(13,17)
```

MEM2SPRITE

Type : Create a sprite from data

Syntax : `result% = MEM2SPRITE(<pixels%[]>,<sprite%>,<width%>,<height%>)`

See Also : SPRITE2MEM

Notes :

- This command returns **TRUE** if the command completed successfully. If it did not, **FALSE** is returned

Result : This command creates a sprite from an array of integers, instead of from a file.

pixels%[] contains the graphic information (in the format of 0xAABBGGRR), where A is the alpha value, B is the blue value, G is the green value and R is the red value.

sprite% is the sprite number to use, whilst **width%** and **height%** is the width and height of the sprite, and determines how many rows and columns there are in **pixels%[]**

Example :

```
// A Tree image
image$ = "" _
        + "....****...." _
        + "..**++++**.." _
        + ".*++++++++*." _
        + "*++++++++++*" _
        + "*++++++*+++*" _
        + "*++++++*+++*" _
        + "*+++++*++++*" _
        + ".*++*+++++*." _
        + "..********.." _
        + "....####...." _
        + "....####...." _
        + "...######..." _
        + ""
DIM colors$[4]
colors$[0] = "."+INTEGER(0x00000000) // transparent
colors$[1] = "+"+INTEGER(0xff00ff80) // green
colors$[2] = "*"+INTEGER(0xff008000) // dark green
colors$[3] = "#"+INTEGER(0xff0080ff) // orange
MakeImage(0, 12, image$, colors$[])

LOCAL pixl%[]
SPRITE2MEM(pixl%[], 0) // get the sprite
LOCAL w%, h%
GETSPRITESIZE 0, w,h
MEM2SPRITE(pixl[], 1, w,h) // make a copy to see if it worked

DRAWRECT 0,0,1000,1000,RGB(70,90,255)
SMOOTHSHADING FALSE
STRETCHSPRITE 0, 32,32,99,99
STRETCHSPRITE 1, 232,32,99,99
SHOWSCREEN
MOUSEWAIT
// ------------------------------------------------------------ //
// -=# MAKEIMAGE #=-
// Make a Sprite from a string
// usage:
```

```
// img$ = ".+." _
// + "+++" _
// ".+."
// dim col$[2];
// col$[0]="."++INTEGER(0xAABBGGRR) // alpha, blue, green, red
// col$[1]="+"++INTEGER(0xff00ff80)
// ------------------------------------------------------------ //
FUNCTION MakeImage: num%, width%, image$, colors$[]
LOCAL x%, y%, c$, height%, pix%[]
   height = LEN(image$)/width
    DIM pix%[width*height]

    FOR y = 0 TO height-1
        FOR x = 0 TO width-1
            c$ = MID$(image$, x+y*width, 1)
            FOR i=0 TO BOUNDS(colors$[], 0)-1
                IF c$ = MID$(colors$[i], 0,1)
                    pix[x + y*width] = MID$(colors$[i], 1, 9)
                    BREAK
                ENDIF
            NEXT
        NEXT
    NEXT

    // Make Sprite from the image
    MEM2SPRITE(pix[], num%, width%, height%)
ENDFUNCTION // MAKEIMAGE
```

MID$

Type : Extracts one or more characters from a string

Syntax : `result$ = MID$(<text$>, start% [, <length%>])`

See Also : LEFT$, RIGHT$

Notes : None

Result : This command returns a section from the text held in **text$**, starting at position **start**, and for the length of characters defined in **length**.

If a value for **length** is not given, or it is -1, then all text up to to the end of **text$** will be used.
The position within **text$** starts at 0

Example :

```
// This example would get 4 characters, starting from position 1, from the text
// "THIS IS A TEST" - which would be 'HIS '
LOCAL A$
A$ = MID$("THIS IS A TEST", 1, 4)
PRINT A$,10,10
SHOWSCREEN
KEYWAIT
OUTPUT: HIS

// Using -1 as the length will return the text from position 1 up to the last character in the
LOCAL A$
A$ = MID$("THIS IS A TEST", 1, -1)
PRINT A$,10,10
SHOWSCREEN
KEYWAIT
OUTPUT: HIS IS A TEST

// Leaving the length parameter blank will return the text from position 1 up to the last character
LOCAL A$
A$ = MID$("THIS IS A TEST", 1)
PRINT A$,10,10
SHOWSCREEN
KEYWAIT
OUTPUT: HIS IS A TEST

// MID$() can be used to perform tasks on strings, in this case
// we copy each letter from the Name$ string to the Reversed$ string
// starting with the last letter and working our way down to the first letter
LOCAL Name$ = "alexander"
LOCAL X
LOCAL Reversed$ // the string that we copy each letter to

// Instead of increasing the value in the FOR loop we decrease it
// going from the last letter to the first
FOR X = LEN(Name$ -1) TO 0 STEP -1
        Reversed$ = Reversed$ + MID$(Name$, X, 1)
NEXT
        PRINT Name$ + " reversed is " + Reversed$, 10, 10
        SHOWSCREEN
KEYWAIT
OUTPUT: alexander reversed is rednaxela
```

MIN

Type : Returns the smaller of two values

Syntax : `result = MIN(<value1>, <value2>)`

See Also : MAX

Notes : None

Result : Returns the smaller value of either **value1** or **value2**

Example :

```
// This sets A to be the smaller of the two values, which is 17
A = MIN(17,24)
```

MOD

Type : Returns the integer part of a division

Syntax : `result% = MOD(<value>, <div>)`

See Also : FMOD

Notes : None

Result : Returns the integer result of **value** divided by **div**

Example :

```
// This sets A to be 0
A = MOD(12, 3)
```

MOUSEAXIS

Type : Retrieve information about mouse movement and button press

Syntax : `result = MOUSEAXIS(<section%>)`

See Also : None

Notes : None

Result : Returns the status of mouse movement or button press, depending on the value of **section** :

Value of **section%**	Information retrieved :
0	X (horizontal) velocity
1	Y (vertical) velocity
2	Wheel movement (< 0 is down) (> 0 is up) (0 is no movement)
3	Left mouse button
4	Middle mouse button
5	Right mouse button

Example :

```
// This example gets the current horizontal velocity
VALUE = MOUSEAXIS(0)
```

MOUSESTATE

Type : Retrieve selected mouse information

Syntax : MOUSESTATE <mx%>, <my%>, <ma%>, <mb%>

See Also : SETMOUSE

Notes : None

Result : Stores the mouse X and Y coordinates into the variables **mx%** and **my%**.

The left and right mouse button states are stored in **ma%** and **mb%** respectively.

If a button is pressed ma% or mb% will equal **TRUE**, otherwise they will equal **FALSE**

Example :

```
// This example gets the current mouse state and stores them in the variables x, y, b1 and b2
MOUSESTATE x, y, b1, b2

////////////////////////////////////////////////////////
// A MOUSESTATE example
// Draws a mouse icon on screen at mouse x and y coordinates
// Checks for mouse button presses
// Turns Green if Left Mouse Button is pressed
// Turns Red if Right Mouse Button is pressed
////////////////////////////////////////////////////////
SETSCREEN 800,600,1
// Variables to store the mouse location and button states
// "mx" is the x position and “my” is the y position of the mouse on screen
// "mba" is the current state of the left mouse button,  Returns 1 if pressed 0 if not pressed
// "mbb" is the current state of the right mouse button, Returns 1 if pressed 0 if not pressed
GLOBAL mx, my, mba, mbb
// Main loop, Repeat until ESCAPE key is pressed
WHILE KEY(1) <> 1
        DrawMouse()
        SHOWSCREEN
WEND
FUNCTION DrawMouse:
        LOCAL R, G, B  // variables to hold the RGB colours
        // Retrieve the mouse coordinates and button states
        MOUSESTATE mx, my, mba, mbb
        // If the left mouse button is pressed
        // change icon to green
        IF mba = 1
                R = 64; G = 255; B = 64
        ELSEIF mbb = 1 // Is Right Mouse button clicked
                R = 255; G = 0; B = 0 // Change to Red
        ELSE
                R = 255; G = 255; B = 255 // Otherwise just display mouse icon as white
        ENDIF

        // Create a simple mouse icon using the line command and draw it using the mouse coordinates
        // mx and my
        DRAWLINE mx, my, mx + 16, my + 10, RGB(R,G,B)
        DRAWLINE mx, my, mx + 10, my + 20, RGB(R,G,B)
        DRAWLINE mx+ 10, my + 20, mx + 10, my + 10, RGB(R,G,B)
        DRAWLINE mx+ 10, my + 10, mx + 16, my + 10, RGB(R,G,B)
        // Output the coordinates next to the mouse icon
        PRINT "X: " + mx + " Y: " + my, mx + 14, my + 20
ENDFUNCTION
```

MOUSEWAIT

Type : Wait for mouse press

Syntax : MOUSEWAIT

See Also : KEYWAIT

Notes : None

Result : The running program is paused until the user presses a mouse button

Example :

```
// This examples makes the program wait until the user presses a mouse button
MOUSEWAIT
```

MUSICVOLUME

Type : Set music volume

Syntax : MUSICVOLUME <volume>

See Also : ISMUSICPLAYING, PLAYMUSIC

Notes : None

Result : Sets the volume for music currently being played. The volume level is set by volume, and is in the range of 0.0 to 1.0, with 0.0 being silent and 1.0 being maximum volume.

Example :

```
// This example sets music volume to maximum
MUSICVOLUME 1.0
```

NETALLOWJOINING

Type : Allow or disallow computers to join servers

Syntax : NETALLOWJOINING <type%>

See Also : None

Notes : None

Result : This command allows the server to allow (**type%** = **TRUE**) client machines to join, or to disallow (**type%** = **FALSE**) clients to join

Example :

```
// This example allows clients to join the server
NETALLOWJOINING TRUE
```

NETCREATEPLAYER

Type : Create a network player

Syntax : id% = NETCREATEPLAYER(<name$>)

See Also : NETDESTROYPLAYER

Notes : None

Result : Creates a player that gets announced to the network server. The network services use "players" for communicating with the server and with other clients.

name$ is the name of the player.

The player ID is stored in **id%**

Example :

```
// This example creates a network player
id%=NETCREATEPLAYER("A Player")
```

NETDESTROYPLAYER

Type : Destroy a player connection to a server

Syntax : NETDESTROYPLAYER <id%>

See Also : NETCREATEPLAYER

Notes : None

Result : This command removes a player from the server. The player ID is stored in **id%**

Example :

```
// This example destroys the player with an ID of 0
NETDESTROYPLAYER 0
```

NETGETIP$

Type : Get list of IP address used by computer

Syntax : result$ = NETGETIP$()

See Also : SOCK_GETIP(), SOCK_GETIP$()

Notes : None

Result : This command returns a list of IP addresses used by the computer. Each IP address is separated by "|"

Example :

```
// This gets the IP address(s) of the IP address for this machine
LIST$ = NETGETIP$()
```

NETGETLASTERROR$

Type : Get error number and description of last error

Syntax : `result$= NETGETLASTERROR$()`

See Also : None

Notes : None

Result : This command returns the error number and description of the last error.

To convert the error message to an integer, you need to use INTEGER(result$)

If no error has occurred, the result is “0 The operation completed successfully.”

Example :

```
// This example returns the last error message
RESULT$ = NETGETLASTERROR$()
```

NETGETMSG$

Type : Reads a network message

Syntax : `result$ = NETGETMSG$(id%)`

See Also : None

Notes : None

Result : This command retrieves a network message for the player defined by **id%**.

If no message has been received, an empty string is returned.

Example :

```
// This example receives a message from the player with an ID of  0
RESULT$ = NETGETMSG$(0)
```

NETGETPLAYERID

Type : Returns ID of a player

Syntax : `result% = NETGETPLAYERID(<index%>)`

See Also : `NETNUMPLAYERS, NETCREATEPLAYER`

Notes : None

Result : This command returns the ID of the player defined by **index%**. The range is between 0 and `NETNUMPLAYERS()` - 1

Example :

```
// This example gets the ID of player 0
A% = NETGETPLAYERID(0)
```

NETGETQUITPLAYER

Type : Returns ID of a player who has quit

Syntax : `result% = NETGETQUITPLAYER()`

See Also : `NETNUMPLAYERS, NETCREATEPLAYER, NETGETPLAYERID, NETSHUTDOWN`

Notes : None

Result : This command returns the ID of a player who has either become disconnected or has been removed by `NETDELETEPLAYER` command.

If no player has quit, this function returns 0

This command only works server side - if a player has quit, then the server should send a message to remaining clients informing them about the loss.

Example :

```
// This example returns the ID of the last player to quit from the server
A% = NETGETQUITPLAYER()
```

NETGETSENDER

Type : Returns ID of the computer that last sent a message

Syntax : `result% = NETGETSENDER()`

See Also : `SOCK_GETREMOTEIP, SOCK_GETREMOTEIP`

Notes : None

Result : This command returns the ID number of the computer that last sent a message.

If no message has been sent, this command returns 0

Example :

```
// This example returns the player that last sent a message
RESULT% = NETGETSENDER()
```

NETHOSTGAME

Type : Host a network game

Syntax : `result%=NETHOSTGAME(<port%>)`

See Also : `NETJOINGAME`

Notes : None

Result : Host a network game using the port number defined by **port%**.

If **port%** = 0, then the port number 27910 is used

If the command completed successfully, then this command returns **TRUE**. If there was a problem, **FALSE** is returned instead

Example :

```
// This example hosts a network game on port 1024
NETHOSTGAME(1024)
```

NETISACTIVE

Type : Determine whether the network system is running

Syntax : `result% = NETISACTIVE()`

See Also : NETCREATEPLAYER

Notes : None

Result : This command returns **TRUE** is the network system is running, or **FALSE** if it is not.

Example :

```
// This example sets A% to TRUE if the network system is running
A% = NETISACTIVE()
```

NETJOINGAME

Type : Join a network game

Syntax : `NETJOINGAME <server$>, <port%>`

See Also : NETHOSTGAME

Notes : None

Result : Join a network game. The IP address of the server is stored in **server$**. The port address to be used is stored in **port%**.

If **port%** = 0, then the port number 27910 is used

Example :

```
// This example joins a host computer using port 1024
NETJOINGAME “192.168.1.1”,1024
```

NETNUMPLAYERS

Type : Determines how many players the server is aware of

Syntax : `result% = NETNUMPLAYERS()`

See Also : NETHOSTGAME

Notes : None

Result : This command returns the number of players that the server is aware of.

Example :

```
// This example stores the number of clients in A
A = NETNUMPLAYERS()
```

NETPLAYERNAME$

Type : Returns display name of a player

Syntax : `result$ = NETPLAYERNAME$(<id%>)`

See Also : None

Notes : None

Result : This command returns the display name of a player. The player chosen is defined by **id%**

Example :

```
// This gets the name of player 0
NAME$ = NETPLAYERNAME$(0)
```

NETSENDMSG

Type : Send a network message

Syntax : `result% = NETSENDMSG(<from%>, <to%>, <text$>)`

See Also : NETGETMSG$, NETGETSENDER

Notes : None

Result : Send a network message from one computer to another.

The sending machine is defined by **from%**, and the received machine is defined by **to%**.

The message to send is defined by **text$**.

If **to%** = 0, then the message is send to all machines, except the sending machine.

The maximum length of **text$** is 1023 characters.

Example :

```
// This example sends a message from the computer with an ID of 0, to all other machines.
RESULT = NETSENDMSG(1, 0, “MESSAGE”)
```

NETSHUTDOWN

Type : Shut-down network system

Syntax : NETSHUTDOWN

See Also : SOCK_SHUTDOWN

Notes : None

Result : Shut down networking system, closes all connections and release all memory used

Example :

```
// This example shuts down the networking system
NETSHUTDOWN
```

NETWEBEND

Type : End a program after launching browser with a URL

Syntax : NETWEBEND <url$>

See Also : SHELLEND

Notes :

- Default web browser is used

Result : This command calls the default web browser for the platform, and uses that to display the web address specified by **url$**

Example :

```
// This example launches a web browser to display the website at www.glbasic.com and then ends
// the program
NETWEBEND "www.glbasic.com"
```

NETWEBGET

Type : Retrieves the contents of a HTTP URL and stores it in a file

Syntax :
```
result% = NETWEBGET(<server$>, <path$>, <port%>,
                    <fileName$>, <timeout%=5000>)
```

See Also : NETWEBGET$

Notes :

- The website address should not start with “http://”, and it should not end with a “/”
- The path text should start with a “/”
- fileName$ should not contain a drive letter
- This command may affect socket connections when used multiple times

Result : This command retrieves the contents of the website defined by **server$**, and at the location within it is defined in **path$**.

The port number to use is defined by **port%**, and is usually 80.

The contents of the website are stored in the filename defined by **fileName$**

timeout% specifies the time (in milliseconds) before the command exits if a connection could not be made.

This command returns **TRUE** if the website exists, and **FALSE** if it doesn’t. This is regardless of whether the rest of the parameters are valid.

Example :

```
// This example gets the INDEX.HTML file from the website WWW.GLBASIC.COM and stores it
// in STORE.TXT
RESULT% = NETWEBGET(“WWW.GLBASIC.COM”, ”/INDEX.HTML”, 80,”STORE.TXT”)
```

NETWEBGET$

Type : Retrieves the contents of a HTTP URL and stores it in a string variable

Syntax :
```
result$ = NETWEBGET$(<server$>, <path$>, <port%>,
                     <length%>, <timeout%=5000>)
```

See Also : NETWEBGET

Notes :

- The website address should not start with “http://”, and it should not end with a “/”
- The path text should start with a “/”
- This command may affect socket connections when used multiple times

Result : This command retrieves the contents of the website defined by **server$**, and at the location within it is defined in **path$**.

The port number to use is defined by **port%**, and is usually 80.

The number of characters that will be returned by this command are defined by **length%**.

timeout% specifies the time (in milliseconds) before the command exits if a connection could not be made.

Example :

```
// This example reads the first 1024 bytes of the INDEX.HTML file from the
// WWW.GLBASIC.COM web site
RESULT$ = NETWEBGET$(“WWW.GLBASIC.COM”, ”/INDEX.HTML”,80,1024)
```

NEXT

Type : Denotes the end of a FOR loop

Syntax : NEXT

See Also : FOR, FOREACH

Notes :
- No variable name is needed after the NEXT command

Result : Denotes the end of a FOR/FOREACH loop. Without this, FOR/FOREACH loops could run into each other causing compiler problems

Example :

```
// This is the basic usage
NEXT
```

Also see the FOR example

NOT

Type : Invert result of logical expression

Syntax : result = NOT(<expression>)

See Also : AND, OR, IF

Notes :
- This command wont invert the sign of a number, instead 0 is always returned

Result : This command returns the opposite result of the **expression**, so if expression evaluates to **TRUE**, NOT would change it to **FALSE**. Likewise if expression evaluates to **FALSE**, NOT would change it to **TRUE**.

This command is usually used with the IF command to invert the result.

Example :

```
// If A$="TEST", then the message is not displayed, otherwise it is displayed
IF NOT(A$="TEST") THEN DEBUG "A$ <> TEST"
```

ON_DEBUG

Type : Interacts with the debugger.

Syntax : N/A

See Also : N/A

Notes : N/A

Result : Not to be used in your programs. This command is only added for completeness.

OPENFILE

Type : Opens a file for reading, writing and appending

Syntax : `result% = OPENFILE(<channel%>, <fileName$>, <mode%>)`

See Also : `CLOSEFILE`

Notes : None

Result : Opens the file defined by **fileName$**, and assigns the channel index **channel** to it. How the file is treated is defined by **mode** :

Value of **mode**	Result
-1	The file is appended
0	The file is written to
1	The file is read from

Example :

```
// This example opens the INFO.TXT file for writing
RESULT = OPENFILE(1, “INFO.TXT”, 0)
```

OR

Type : Logical operator

Syntax : `result = <value1> OR <value2>`

See Also : AND, NOT

Notes : None

Result : Returns **TRUE** if either **value1** OR **value2** are non-zero

Example :

```
// A is TRUE if B or C are non-zero
A = B OR C
```

PAUSEMUSIC

Type : Pause or continue playing music

Syntax : PAUSEMUSIC <pause%>

See Also : PLAYMUSIC, STOPMUSIC

Notes : None

Result : Pauses (**pause%** = **TRUE**) or continue playing (**pause%** = **FALSE**) music

Example :

```
// Pause music that is playing
PAUSEMUSIC TRUE
```

PLATFORMINFO$

Type : Returns platform specific information

Syntax : `result$ = PLATFORMINFO$(<type$>)`

See Also : None

Notes :

- Platforms that don't use battery power will return “100” if **type$** = “BATTERY”
- Currently using a type of “DEVICE” or “BATTERY” will return “UNKNOWN” and “NO_DATA” respectively

Result : Platform specific information, based on the value of **type$**. It can be any of the following valid values :

Value of **type$**	Result
“” (empty string)	The platform the program is running on. This can be : • LINUX - Linux platform • WIN32 - Windows platform • WINCE - Windows CE platform • GP2X - GP32 platform • MACOSX - Mac OS platform • IPHONE - iPhone or iPod Touch • PANDORA – Pandora • WEBOS – Palm Pre/Palm Pre 2 • ANDROID – Android platforms
“ID”	Unique computer ID
“DEVICE”	What hardware the program is running on. This can be : • DESKTOP - A desktop machine • POCKETPC - A WinCE touch screen machine • SMARTPHONE - A WinCE with no touch screen • WINCE - A WinCE machine but with no further information • F100 - A GP2X F-100 machine • F200 - A GP2X F-200 machine, with touch screen

	• IPHONE - iPhone • IPOD - iPod • IPOD TOUCH - iPod Touch (V8 onwards) • IPAD - iPad (V8 onwards)
“BATTERY”	The battery power remaining in percentage
“TIME”	Returns the current time in the format of year-month-date hour:minute:second
“GLEXT:xxxx”	Returns whether the GL extension of “xxxx” has been found. If it has then TRUE will be returned, otherwise FALSE will be
“COMPILED”	Returns the date and time that the program was compiled, using the same layout as the TIME option.
“HOSTID”	Returns the serial number of the computer that the program was compiled on
“LOCALE”	Returns the operating system locale in lower-case letters, based on the NASA country codes
“DOCUMENTS”	Returns the path to the users ‘Documents’ folder - for some platforms, this is the only place where files can be stored.
“TEMP”	Returns the path to the temporary directory (V8 onwards)
“VERSION”	Returns the version string as set in Project → Options. The version string is in the format of version.revision (xxx.yyy). For example : 001.001
“APPDATA”	Returns a directory to put application specific settings. A sub-directory with the programs's name should be put here

Example :

```
// This example gets the computer ID
RESULT$ = PLATFORMINFO$(“ID”)
```

PLAYMOVIE

Type : Plays a movie

Syntax : PLAYMOVIE <fileName$>

See Also : LOOPMOVIE

Notes :

- This command will only work with Windows platforms

Result : Loads and plays the movie file **fileName$**

Example :

```
// This plays the movie 'MOVIE.AVI' once
PLAYMOVIE "MOVIE.AVI"
```

PLAYMUSIC

Type : Start playing some music

Syntax : PLAYMUSIC <fileName$>, <loop%>

See Also : STOPMUSIC, MUSICVOLUME

Notes :

- Each platform expects music to be in different formats/baud rates, so a MP3 file played on a Windows machine may not work on a Palm Pre, unless the format is changed.

Result : Starts playing some music, without interrupting the running program. If **loop% = TRUE**, then the music will restart playing once it has finished. If **loop% = FALSE**, then the music will only play once

The filename of the music to be played is in **fileName$**

Example :

```
// This example plays the MP3 called MUSIC once
PLAYMUSIC "MUSIC.MP3", FALSE
```

PLAYSOUND

Type : Plays a loaded sound

Syntax : `channel% = PLAYSOUND(<index%>, <pan>, <volume>)`

See Also : LOADSOUND, SOUNDPLAYING

Notes : None

Result : Starts playing the sound that was loaded into sound index **index%**. The sound can be changed to appear from coming out from the left and/or right speakers by changing the value held in **pan**, according to the following ranges :

Value of **pan**	Effect
-1	Sound comes from left speaker
0	Sound comes from left and right speakers
1	Sound comes from right speaker

The volume level for the sound is set by **volume**, with 0.0 being silent and 1.0 being maximum volume.

The value returned is the channel number being used to play the sound.

Example :

```
// This example plays the sound from index 1, with the sound coming out of left and right speakers, and
with maximum volume
CHANNEL% = PLAYSOUND(1, 0.0,1.0)
```

POLYNEWSTRIP

Type : Inserts a de-generated triangle in "strip" mode

Syntax : POLYNEWSTRIP

See Also : STARTPOLY, ENDPOLY

Notes : None

Result : This command allows drawing of polygons without needing multiple STARTPOLY/ENDPOLY commands, allowing graphics display to be sped up.

Example :

```
STARTPOLY 0,2

LOCAL c%=RGB(255,255,255)
POLYVECTOR 0, 0, 0,0,c
POLYVECTOR 0,40, 0,0,c
POLYVECTOR 40,20, 0,0,c
POLYVECTOR 40,40, 0,0,c

POLYNEWSTRIP

// same at 100,100
POLYVECTOR 100,100, 0,0,c
POLYVECTOR 100,140, 0,0,c
POLYVECTOR 140,120, 0,0,c
POLYVECTOR 140,140, 0,0,c

ENDPOLY
SHOWSCREEN
MOUSEWAIT
```

POLYVECTOR

Type : Define a polygon

Syntax : POLYVECTOR <x%>, <y%>, <tx%>, <ty%>, <colour%>

See Also : STARTPOLY, ENDPOLY, POLYNEWSTRIP

Notes : None

Result : Used in conjunction with STARTPOLY and ENDPOLY, this command defines a polygon at the coordinates given by **x%** and **y%**.

The textures pixels are given by the coordinates held in **tx%** and **ty%**

The colour given this part of the polygon is defined by **colour%**.

There needs to be a minimum of 3 POLYVECTOR commands within the STARTPOLY and ENDPOLY sections, otherwise nothing is drawn

Example :

```
// This example creates a poly-vector starting at 1,1 and with a width and height of 8 pixels.  It
// has a purple colour
STARTPOLY 500
   POLYVECT 1,1, 8, 8,RGB(255,0,255)
ENDPOLY
```

POW

Type : Returns the result of a value raised to a power

Syntax : result = POW(value,power)

See Also : None

Notes : None

Result : Returns the result of **value** raised to the power of **power**.

Example :

```
// This sets A to be 32
A = POW(2,5)
```

PRINT

Type : Puts text onto the back buffer

Syntax : `PRINT text$, x, y [, bKernal% = FALSE]`

See Also : `SETFONT, LEN`

Notes :

- If no font is loaded, the default character set is very small and hard to read
- Default font files (smalfont.png) should go in the same place as the executable, although on a Mac, it is usually easier to load from the "media" directory and load then manually
- Font kernelling wont work with the default character set

Result : Puts the text defined by **text$** onto the back buffer at the coordinates defined by **x** and **y**, using the current font.

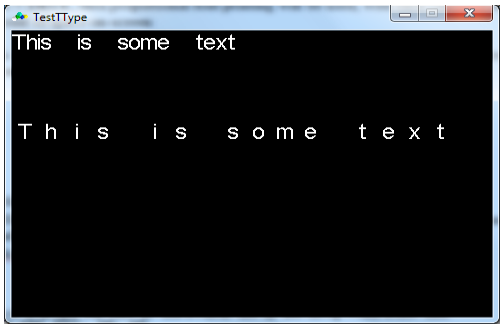
If **bKernal%** = **TRUE**, then proportional font printing will be used, with each character taking up a different amount of space on-screen

If **bKernal%** = **FALSE**, then fixed-width printing will be used. This means each character will take up the same amount of space.

Example :

```
// This example displays the text at coordinates 100 by 100 using fixed-width text
PRINT "THIS IS SOME TEXT", 100, 100

// This prints the same text with proportional printing
PRINT "THIS IS SOME TEXT", 100, 100, TRUE
```

The different between fixed-width and proportional font printing can be seen in this picture :



The top row is proportional text, whilst the bottom row is fixed-width. Notice how much extra space the latter takes up.

PROTOTYPE

Type : Allows a function address to be passed as a parameter

Syntax : PROTOTYPE name | name% | name$ | name as *type* :
[param1, param2, ..paramN]

See Also : FUNCTION, ADDRESSOF

Notes :

- This command cant be used to point to function in an extended type

Result : Working somewhat like ADDRESSOF (but not limited to working with only one command), this command allows a function address to be stored in a variable, and then called from that variable.

This allows the creation of jump tables.

name is the name of the function that is to be prototypes. **param1** to **paramN** are the parameters it takes (if any)

Example :

```
// This example defines the function to be prototyped
PROTOTYPE func_proto: a
TYPE Tfoo
   moo AS func_proto
ENDTYPE
LOCAL pFoo AS func_proto
// This is a check to see if the variable pFoo has been assigned a function
IF pFoo=FALSE THEN STDOUT "has no pointer\n"
pFoo = f1 // Get the address of the  function called “f1”
pFoo(1) // And call it

// Get the address of the function call “f2”
pFoo = f2; pFoo(2) // And call it
call(pFoo)

LOCAL tp AS Tfoo
tp.moo = f1; tp.moo(4); ALIAS fkt AS tp.moo; fkt(5); KEYWAIT

FUNCTION f1: one
  STDOUT "f1: "+one+"\n"
ENDFUNCTION

FUNCTION f2: a
  STDOUT "f2: "+a+"\n"
ENDFUNCTION

FUNCTION call: foo AS func_proto
  foo(3)
ENDFUNCTION
```

PUTENV

Type : Set environment variable to a value

Syntax : PUTENV <variable$>, <value$>

See Also : GETENV$

Notes : None

Result : This command sets the environment variable held in **variable$** with the value held in **value$**.

This value is usually only retained whilst the program is running, but will be inherited by any child processes (for example when called using SHELLCMD)

Example :

```
// This command will set the environment "HOME" to "C:/"
PUTENV "HOME","C:/"
```

PUTFILE

Type : Writes data to a specified line in a file

Syntax : PUTFILE <fileName$>, <index%>, <data$>

See Also : GETFILE

Notes : None

Result : Writes the data held by **data$** into the file defined by **fileName$** and at the given line number defined by **index%**

Example :

```
// This example writes DATA into the 4th line in the RESULT.TXT file
PUTFILE "RESULT.TXT",4,"DATA"
```

READ

Type : Read item of data

Syntax : READ <data> [, <data>, ...]

See Also : STARTDATA, ENDDATA, RESTORE

Notes :

- If RESTORE has NOT been used before this command is used, the compiler will generate an "Out of data" error message. This error message will also be generated when trying to read more data than that present.

Result : Read an item of data from the block of data that was defined by RESTORE. The value read should match the variable type that **data** will be stored in.

Example :

```
// This will allow the data block called "info" to be read from
RESTORE INFO

// This will read the data in.  The variable A wil hold 1, B will hold 2, C
// will hold 3 and D$ will hold the text "FOUR".
READ A,B,C,D$

// To use data, it must be surrounded with STARTDATA and ENDDATA
STARTDATA INFO:
  DATA 1,2,3,"FOUR"
ENDDATA
```

READBYTE

Type : Reads a signed-byte from an opened file

Syntax : READBYTE <handle%>, <store%>

See Also : READUBYTE

Notes : None

Result : Reads a byte (as a value between -127 and +127) from the opened file held in **handle%**, and stores the value in **store%**

Example :

```
// This reads a signed byte from a file and stores it in A%
READBYTE 1, A%
```

READIEEE

Type : Reads a double-precision floating-point value from an opened file

Syntax : READIEEE <handle%>, <store>

See Also : READSHORTIEEE

Notes :
- If eight bytes can not be read, your program will crash

Result : Reads a double-precision (8 byte) floating-point value from the opened file held in **handle%**, and stores the value in **store**

Example :

```
// This reads a double float from a file and stores it in A
READIEEE 1, A
```

READLINE

Type : Reads data until a line terminator is found

Syntax : READLINE <handle%>, <store$>

See Also : WRITELINE

Notes : None

Result : Reads in data from the opened file held in **handle%**, and stores them into **store$**.

Data is read until a line terminator ("\n") is found.

Example :

```
// This example reads a line from an open file and stores it in STORE
READLINE 1, STORE$
```

READLONG

Type : Reads a signed-long from an opened file

Syntax : READLONG <handle%>, <store%>

See Also : READULONG

Notes :

- If four bytes can not be read, your program will crash

Result : Reads a signed-long (as a value between -2,147,483,648 and +2,147,483,647) from the opened file held in **handle%**, and stores the value in **store%**

Example :

```
// This reads a signed long from a file and stores it in A%
READLONG 1, A%
```

READSHORTIEEE

Type : Reads a single-precision floating-point value from an opened file

Syntax : READSHORTIEEE <handle%>, <store>

See Also : READIEEE

Notes :

- If four bytes can not be read, your program will crash

Result : Reads a single-precision (4 byte) floating-point value from the opened file held in **handle%**, and stores the value in **store**

Example :

```
// This reads a float from a file and stores it in B
READSHORTIEEE 1, B
```

READSTR

Type : Reads a given number of characters from an opened file

Syntax : READSTR <handle%>, <store$>, <count%>

See Also : WRITESTR

Notes : None

Result : Reads a given number of characters from the opened file held in **handle%**, and stores them into **store$**. The number of characters to read is held in **count%**. If this value is 0, then any text already present in **store$** will be removed.

Example :

```
// This reads 5 bytes from the open file, and stores it in STORE$
READSTR 1, STORE$, 5
```

READUBYTE

Type : Reads an unsigned-byte from an opened file

Syntax : READUBYTE <handle%>, <store%>

See Also : READBYTE

Notes : None

Result : Reads a byte (as a value between 0 and 255) from the opened file held in **handle%**, and stores the value in **store%**

Example :

```
// This reads an unsigned byte from a file and stores it in B%
READUBYTE 1, B%
```

READULONG

Type : Reads an unsigned-long from an opened file

Syntax : READULONG <handle%>, <store%>

See Also : READLONG

Notes :

- If four bytes can not be read, your program will crash

Result : Reads an unsigned-long (as a value between 0 and +4,294,967,295) from the opened file held in **handle%**, and stores the value in **store%**

Example :

```
// This reads an unsigned long from a file and stores it in A%
READULONG 1, A%
```

READUWORD

Type : Reads an unsigned-word from an opened file

Syntax : READUWORD <handle%>, <store%>

See Also : READWORD

Notes :

- If two bytes can not be read, your program will crash

Result : Reads an unsigned-word (as a value between 0 and +65535) from the opened file held in **handle%**, and stores the value in **store%**

Example :

```
// This reads an unsigned word from a file and stores it in A%
READUWORD 1, A%
```

READWORD

Type : Reads a signed word from an opened file

Syntax : READWORD <handle%>, <store%>

See Also : READUWORD

Notes :
- If two bytes can not be read, your program will crash

Result : Reads a signed-word (as a value between -32767 and +32767) from the opened file held in **handle%**, and stores the value in **store%**

Example :

```
// This reads a signed word from a file and stores it in A%
READWORD 1, A%
```

REDIM

Type : Resizes an array

Syntax : REDIM <var>[size1]*[[size2]...[size4]]*

See Also : DIM

Notes : None

Result : Re-sizes the array variable **var**, which contains **size1** elements. Multi-dimensional arrays are allowed and are given one after the other (**size2** to **size4**)

Using REDIM **var**[0] will not de-allocate memory and will allow quicker re-dimensioning of a once-single dimensioned array

Example :

```
// This example changes the size of the VAR array to have 12 elements
REDIM VAR[12]
```

REPEAT

Type : Start a loop that will run at least once

Syntax : REPEAT

See Also : UNTIL

Notes : None

Result : Start a conditional loop that will be executed 1 or more times.

Example :

```
// REPEAT usage
REPEAT

// Loop until A > 10
REPEAT
    ...
UNTIL A>10
```

REPLACE$

Type : Replace text

Syntax : result$ = REPLACE$(<text$>, <find$>, <replace$>)

See Also : INSTR

Notes : None

Result : This command searches the contents of **text$**, looking for all occurrences of the value held in **find$**. If any are found, then they are changed to the contents of **replace$**

Example :

```
// This example replaces the word 'IS A' with 'WAS A' to get 'THIS WAS A TEST'
A$ = REPLACE$("THIS IS A TEST",A1"IS A", "WAS A")
```

REQUIRE

Type : Include a C/C++ source file into the program

Syntax : REQUIRE <fileName$>

See Also : INLINE, IMPORT

Notes : None

Result : Include a C/C++ source file into the program during compilation. The file included is stored in **fileName$**

Example :

```
// This example include the C++ file "code" into the current program
REQUIRE "CODE.CPP"
```

RESTORE

Type : Set which data block to read

Syntax : RESTORE <label>

See Also : STARTDATA, ENDDATA, READ

Notes :

- If **label** doesn't exist, the compiler will generate a "missing DATASTART for label :" error message

Result : Define which block of data can be read next. The name of the block of data is defined by **label**

Example :

```
// This will allow the data block called "info" to be read from
RESTORE info

// To use data, it must be surrounded with STARTDATA and ENDDATA
STARTDATA info:
    DATA 1,2,3,"FOUR",5,6
ENDDATA
```

RETURN

Type : Return back to the statement following the SUB/FUNCTION call

Syntax : RETURN *[<var>]*

See Also : SUB, FUNCTION

Notes :

- If **var** is not present, then 0 is returned

Result : Returns the value held in **var** back to the caller.

Example :

```
// This is how the command is used
RETURN 12

// This example shows how a value is returned from a function
FUNCTION test:
   RETURN 12
ENDFUNCTION
```

REVINSTR

Type : Reverse search text within a string

Syntax : `pos%=REVINSTR(<string$>, <subString$> [, <start%=-1>])`

See Also : `INSTR`

Notes : None

Result : This command looks for the string held in **subString$**, within the text held by **string$**. The starting position is defined by the value held in **start%**. If this is -1, then the search starts from the end of the string held in **string$** to the beginning.

If the starting value is not -1, then the search starts from value given from the start of the string.

The command returns the first occurrence of the text held by **subString$** or -1 if it wasn't found.

Example :

```
LOCAL haystack$ = "the quick brown fox jumps over the lazy dog"
LOCAL pos%

//  Returns 31
pos = REVINSTR(haystack$, "the")
STDOUT "pos1: "+pos+"\n"

// Returns 0
pos = REVINSTR(haystack$, "the", 16)
STDOUT "pos2: "+pos+"\n"

// Returns 41
pos = REVINSTR(haystack$, "g")
STDOUT "g: "+pos+"\n"

KEYWAIT
```

RGB

Type : Combines the colour of red, green and blue into one value

Syntax : `result% = RGB(<red%>, <green%>, <blue%>)`

See Also : None

Notes :

- Up until version 8.125, this was the only command that could be used to initialise variables

Result : This command combines the values held in **red%**, **green%** and **blue%** into one value, for use with commands that require a colour parameter.

The values for **red%**, **green%** and **blue%** can be between 0 and 255

Example :

```
// This example returns the integer value of yellow which in this case is 65535
A = RGB(255, 255, 0)
```

RIGHT$

Type : Extracts a given number of characters from the right-hand side of a string

Syntax : `result$ = RIGHT$(<text$>, <num%>)`

See Also : `LEFT$`, `MID$`

Notes : None

Result : This command returns a given number of characters, defined by **num%**, from the right-hand side of the string defined by **text$**

If **num%** = 0, then an empty string is returned

If **num%** exceeds the number of characters in text$, then the complete contents of **text$** is returned

Example :

```
// This example would display "TEST"
PRINT RIGHT$("THIS IS A TEST",4), 10, 10
SHOWSCREEN
KEYWAIT
```

RND

Type : Generate a pseudo-random number

Syntax : `result = RND(<highest>)`

See Also : SEEDRND

Notes : None

Result : Returns a pseudo-random number between 0 and **highest** (inclusive).

Example :

```
// A can be any value between 0 and 16
A = RND(16)
```

ROTOSPRITE

Type : Draws a rotated sprite

Syntax : `ROTOSPRITE <index%>, <x>, <y>, <angle>`

See Also : DRAWSPRITE

Notes :

- The sprite is rotated around its centre

Result : Draws a sprite from sprite index **index%**.

Horizontal positioning is set by **x**, and vertical positioning is defined by **y**.

The angle to rotate the sprite is defined by **angle**. This should be a value between 0 and 359, although any value is accepted.

Example :

```
// This example rotates sprite index 1 to 90 degrees and positions it at coordinates 100 by 100
ROTOSPRITE 1, 100, 100,90
```

ROTOZOOMANIM

Type : Draws a rotated and scaled animated sprite

Syntax : ROTOZOOMANIM <index%>, <tile%>, <x>, <y>, <angle>, <scale>

See Also : ROTOSPRITE

Notes : None

Result : Draws an animated sprite from index **index%**, and using the tile index **tile%**.

The horizontal position is set by **x**, and the vertical by **y**.

The angle to use is defined by **angle**, and the scale by **scale**.

Example :

```
// This example zooms sprite index 1 and animation frame 2 by 1.2, rotates it by 90 and places it at
// coordinates 100 by 100
ROTOZOOMANIM 1, 2, 100, 100, 90, 1.2
```

ROTOZOOMSPRITE

Type : Draws a rotated and scaled sprite

Syntax : ROTOZOOMSPRITE <index%>, <x>, <y>, <angle>, <scale>

See Also : ZOOMSPRITE, ROTOSPRITE

Notes :

- Rotation is performed around the centre of the sprite

Result : Draws a sprite from index **index%**.

The horizontal position is set by **x,** and the vertical by **y**.

The angle to use is defined by **angle**, and the scale by **scale**.

Example :

```
// This example scales sprite index 1 by 1.4, rotates it by 90 degrees and positions it at
// coordinates 100 by 100
ROTOZOOMSPRITE 1, 100, 100, 90, 1.4
```

RTRIM$

Type : Trims text from the end of text

Syntax : `result$ = RTRIM$(text$ [, remove$="\t\r\n\v\f"])`

See Also : `TRIM$`, `LTRIM$`

Notes : None

Result : This removes all text held in **remove$** from the text in **text$**. Text is removed from the left-hand side of the text only.

Example :

```
// This example removes all "a"'s from the end of the text "abcdefa"
DEBUG RTRIM$("abcdefa","a")
```

SAVEBMP

Type : Saves the back buffer to a file

Syntax : `SAVEBMP <fileName$>`

See Also : None

Notes :

- The extension denotes the format of the file (".bmp" for Windows bitmap and ".png" for PNG bitmap and ".jpg" for JPEG files)

Result : Saves the contents of the back buffer to the file defined by **fileName$**

Example :

```
// This example saves the current back buffer to the file EXAMPLE.BMP
SAVEBMP "EXAMPLE.BMP"
```

SAVESPRITE

Type : Saves a defined sprite to a file

Syntax : `SAVESPRITE <fileName$>, <index%>`

See Also : None

Notes :

- The extension denotes the format of the file (".bmp" for Windows bitmap and ".png" for PNG bitmap and ".jpg" for JPEG files)
- If the sprite index isn't in use then no file is created

Result : Saves the sprite defined by **index%** to the file defined by **fileName$**

Example :

```
// This example save the sprite index 1 to the file OUTPUT.BMP
SAVESPRITE "OUTPUT.BMP", 1
```

SEEDRND

Type : Initialise the pseudo random number generator with a new starting value

Syntax : SEEDRND <value>

See Also : RND

Notes :

- If you find that you get the same numbers repeatedly even with this command, you might need to find a better way of calculating the seed value.

Result : Initialises the random number generator with the starting seed of **value**

Example :

```
// Reseed the random number generator based on the current time since the computer was switched
// on
SEEDRND GETTIMERALL()
```

SELECT

Type : Start complex comparison branching

Syntax : SELECT <var>

See Also : CASE, DEFAULT, ENDSELECT

Notes :

- This command is converted to one or more IF statements during program compiling

Result : Initialise the ability to perform complex comparison branching based on the result of **var**

Example :

```
// The layout of SELECT is as follows
SELECT A
   CASE    // At least one is needed
   CASE
   DEFAULT // This is optional
ENDSELECT

// A more usual use is :
SELECT A
    CASE 1
         // Do something if A = 1
    CASE 2
         // Do something else if A = 2
    CASE > 3
         // Do something if A is greater than 3
    DEFAULT
         // Do something else if nothing above
         // is valid
ENDSELECT
```

SETACTIVEMOUSE

Type : Select a different mouse to receive input from

Syntax : SETACTIVEMOUSE <index%>

See Also : GETMOUSECOUNT

Notes :

- Currently, this command is only valid for the iPhone/iPod/iPad and Palm Pre

Result : Select data to be read from the mouse defined by **index%**

Example :

```
// This example sets mouse 1 to have data read from
SETACTIVEMOUSE 1
```

SETCURRENTDIR

Type : Changes the current working directory

Syntax : SETCURRENTDIR <dir$>

See Also : GETCURRENTDIR$

Notes : None

Result : Sets the working directory to that held by **dir$**

Example :

```
// This changes the current directory to the MOVIE directory
SETCURRENTDIR "../MOVIES"
```

SETFONT

Type : Use a previously loaded font for text display

Syntax : SETFONT <font%>

See Also : LOADFONT

Notes : None

Result : Uses the previously loaded font that is defined by **font%**.

All text will then be rendered using the selected font

Example :

```
// This example sets the current font to be used to index 1
SETFONT 1
```

SETMOUSE

Type : Sets the mouse X and Y positions

Syntax : SETMOUSE <mx%>, <my%>

See Also : MOUSESTATE

Notes : None

Result : Sets the mouse X and Y position to that contained in **mx%** and **my%**

Example :

```
// This example sets the mouse position to an X value of 100, and a Y value of 100
SETMOUSE 100, 100
```

SETORIENTATION

Type : Set rendering orientation

Syntax : `SETORIENTATION iMode%`

See Also : `GETORIENTATION`

Notes : None

Result : Changes the orientation that all rendering is performed at. All mouse and joystick input values are changed too.

The orientation value held in **iMode%** can be one of the following :

Value of **iMode%**	Orientation :
0	Up (Portrait)
1	Right (Landscape)
2	Down (Inverse Portrait)
3	Left (Inverse Landscape)

Example :

```
X_OBJSTART 0
X_OBJADDVERTEX 0, 20, 0, 0,0,0xffffff
X_OBJADDVERTEX 0,-20, 10, 0,0,0xff0000
X_OBJADDVERTEX 0,-20,-10, 0,0,0x0000ff
X_OBJNEWGROUP
X_OBJADDVERTEX 0, 20,0, 0,0,0xffffff
X_OBJADDVERTEX 30,-10,0, 0,0,0xff0000
X_OBJADDVERTEX -30,-10,0, 0,0,0x0000ff
X_OBJNEWGROUP
X_OBJADDVERTEX 10.5,-15, 10.5, 0,0,0x00ffff
X_OBJADDVERTEX 10.5, 15, 10.5, 0,0,0x00ffff
X_OBJADDVERTEX -10.5,-15, -10.5, 0,0,0x00ffff
X_OBJADDVERTEX -10.5, 15, -10.5, 0,0,0x00ffff
X_OBJEND

LOCAL mx%, my%, b1%, b2%
LOCAL irot% = 0
WHILE TRUE

    IF KEY(2) THEN irot=0
    IF KEY(3) THEN irot=1
    IF KEY(4) THEN irot=2
    IF KEY(5) THEN irot=3

    // rotate the screen. Stays locked until you change that!
    SETORIENTATION irot

    // mouse position on screen (in screen pixels - not device)
    MOUSESTATE mx, my, b1, b2

    // print where mouse is and rotation index
    PRINT "Rot: "+irot+"Mouse: "+FORMAT$(4,0,mx)+ ", "+FORMAT$(4,0,my), 0,0
```

```
    // get the full viewport
LOCAL vpx%, vpy%, x%, y%
    GETVIEWPORT x, y, vpx, vpy

    DRAWLINE x, y, vpx, y, RGB(255,128,128)
    DRAWLINE x, y, x, vpy, RGB(255,128,128)
    DRAWLINE x+vpx-1,y, x+vpx-1,vpy, RGB(255,128,128)
    DRAWLINE x, y+vpy-1,vpx, y+vpy-1, RGB(255,128,128)

    PRINT "VP: "+x+","+y+" "+vpx+"x"+vpy, 0,20

    // and screen size
    GETSCREENSIZE vpx, vpy
    PRINT "SC: "+vpx+"x"+vpy, 0,40

    // draw line to viewport rect
    DRAWLINE vpx, vpy, 50+200,100+100, RGB(0,255,255)

    // render to a viewport area
    VIEWPORT 50,100, 200,100
    // make sure we don't paint outside the viewport
    DRAWRECT -999,-999,2000,2000,RGB(64,64,128)

    // render 3D into that viewport
    do3D()


    // start rendering in 2D over the same viewport again
    X_MAKE2D
    VIEWPORT 50,100, 200,100
    PRINT "2D Viewport 200x100",0,0
    // is the size correct?
    GETVIEWPORT x, y, vpx, vpy
    PRINT "VP: "+x+","+y+" "+vpx+"x"+vpy, 0,20


    // now grab that viewport and paste it again
    VIEWPORT 0,0,0,0
    GRABSPRITE 1, 50,100,200,100
    DRAWSPRITE 1, 50,201



    // mouse position
    DRAWRECT mx-4, my-4, 8,8,RGB(255,255,0)


    SHOWSCREEN

WEND

FUNCTION do3D:
    ALPHAMODE 0
    X_CULLMODE 0

    X_MAKE3D 1, 100, 45
    X_CAMERA 3,3,40, 0,0,0
    X_ROTATION GETTIMERALL()*0.1, 0,1,0
    X_DRAWOBJ 0,0
ENDFUNCTION
```

SETPIXEL

Type : Draw a pixel on the back buffer

Syntax : `SETPIXEL <x%>, <y%>, <colour%>`

See Also : `GETPIXEL`

Notes : None

Result : Puts the specified coloured pixel (defined by **colour%**), into the back buffer at the coordinates specified by **x** and **y**.

Example :

```
// This sets the pixel at 100, 100 to red
SETPIXEL 100, 100, RGB(255, 0, 0)
```

SETSCREEN

Type : Set screen resolution

Syntax : SETSCREEN <width%>, <height%>, <fullScreen%>

See Also : GETSCREENSIZE, AUTOPAUSE

Notes :

- The iPhone/iPod, GP2X/WIZ, Windows CE, Palm Pre and Pandora can only use a full screen
- AUTOPAUSE should be used after this command
- Most platforms require all graphics to be re-loaded if screen resolution is changed

Result : Sets up a new graphics mode using the width and height values defined in **width%** and **height %**.

If **fullScreen%** = **TRUE**, then the graphics screen is displayed in full-screen mode.

If **fullScreen%** = **FALSE**, then the graphics screen displayed in a window.

If **width%** and **height%** = 0, then the current screen sizes are used

Example :

```
// This sets the graphics screen to full screen mode and to a resolution of 1024 x 768
SETSCREEN 1024, 768, TRUE
```

SETSHOEBOX

Type : Set files to use for loading compressed data

Syntax : SETSHOEBOX <data$>, <media$>

See Also : None

Notes :

- Shoebox files are created with the "Shoebox" utility in GLBasic

Result : Specifies which shoebox file to be used to load data from, unless it already appears in uncompressed form.

The file which LOADSPRITE, LOADSOUND, LOADFONT, LOADBMP, BLENDSCREEN, GETFILE use is defined by **data$**

The file which PLAYMUSIC, PLAYMOVIE, LOOPMOVIE use is defined by **media$**

Example :

```
// This example defines DATA.SBX as where to find graphics, and MEDIA.SBX as the place for
// music and movies
SETSHOEBOX "DATA.SBX", "MEDIA.SBX"
```

SETSPRITEANIM

Type : Set tile size for sprites

Syntax : SETSPRITEANIM <id%>, <w%>, <h%>

See Also : DRAWANIM, LOADANIM

Notes : None

Result : Sets the tile size if you want to use any sprite with DRAWANIM.

id% denotes which sprite to use, whilst **w%** and **h%** denote the width and height of the tile to be used

Example :

```
LOADANIM "test.png", 100, 32,32

// does the same as:
LOADSPRITE "test.png", 100
SETSPRITEANIM 100, 32,32
```

SETTRANSPARENCY

Type : Defines the colour used to be ignored when displaying graphics

Syntax : SETTRANSPARENCY <colour%>

See Also : None

Notes :

- To create transparency masks easily, and to avoid using this command, PNG files are the way to go!

Result : This command defines what colour the LOADFONT, GRABSPRITE and LOADSPRITE commands see has the transparent colour, and thus ignored when placing graphics, allowing the background to be visible. The colour to be used is defined by **colour%**

The defaults for these commands are as follows :

2D DRAWING COMMANDS	Colour
LOADFONT	RGB(0, 0, 0)
GRABSPRITE	RGB(255, 0 ,128)
LOADSPRITE	RGB(255, 0, 128)

Example :

```
// This sets transparency to black - so any pixels in sprites or fonts that are black wont be
// displayed
SETTRANSPARENCY RGB(0,0,0)
```

SGN

Type : Return sign of a number

Syntax : `result% = SGN(value)`

See Also : ABS

Notes : None

Result : Returns a value based of the sign of **value.** The return value can be :

Value of **value**	Return value
> 0	1
0	0
< 0	-1

Example :

```
// This example sets A to -1 as -12 is less than 0
A% = SGN(-12)
```

SHELLCMD

Type : Execute another program or call a DOS/Terminal command

Syntax : `result% = SHELLCMD(<command$>, <wait%>, <show%>, <result>)`

See Also : SHELLEND

Notes : None

Result : Executes another program , and, for Windows, allows execution of a DOS command.

command$ holds the file-name or DOS command.

wait% specifies whether to wait for the called command to finish (**wait%** = **TRUE**), or continue program execution (**wait%** = **FALSE**).

show% denotes whether to the program is started hidden (**show%** = **FALSE**) or visible (show = **TRUE**)

The return value from the other program is stored in result, and this command returns **TRUE** if the command completed successfully or **FALSE** if there was a problem.

Example :

```
// Calls the NOTEPAD.EXE program, continues executing the main program whilst it is being run
// and returns the error status into result
SHELLCMD("c:/windows/system32/notepad.exe",FALSE,TRUE,vv)
```

SHELLEND

Type : Terminates program execution after calling another program

Syntax : SHELLEND <fileName$>

See Also : SHELLCMD, NETWEBEND

Notes : None

Result : Replaces the currently executed program with that stored in **fileName$**

Example :

```
// Exits the running program and calls NOTEPAD.EXE
SHELLEND "NOTEPAD.EXE"
```

SHOWSCREEN

Type : Swaps the front and back buffers

Syntax : SHOWSCREEN

See Also : CREATESCREEN, SETSCREEN

Notes :

- This command should not be used when compiling in console mode
- This command also resets ALPHAMODE to 0.0 and SMOOTHSHADING to **TRUE**

Result : This command swaps the front and back buffers, showing the contents of the back buffer. This is needed to show everything drawn.

Example :

```
 // This shows the back-buffer
SHOWSCREEN
```

SIN

Type : Returns the sine of a value

Syntax : `result = SIN(<value>)`

See Also : ASIN, ACOS, ATAN, COS, TAN

Notes : None

Result : Returns the sine held in **value**.

Example :

```
// Sets A to be 0.2923717047
A = SIN(17.0)
```

SLEEP

Type : Make the program sleep

Syntax : `SLEEP <time>`

See Also : HIBERNATE

Notes : None

Result : Causes the program to halt (sleep) for a certain number of milliseconds, as defined by **time**

Example :

```
// This example pauses the program for 16 milliseconds
SLEEP 16
```

SMOOTHSHADING

Type : Sets interpolation type for scaling sprites

Syntax : SMOOTHSHADING <mode%>

See Also : X_MAKE2D, SHOWSCREEN

Notes : None

Result : This command allows scaling sprites to be performed by soft interpolation (**mode%** = **TRUE**) or fast interpolation (**mode%** = **FALSE**) when scaling sprites

X_MAKE2D and SHOWSCREEN resets this command to use soft interpolation

Example :

```
// This example enables soft interpolation
SMOOTHSHADING TRUE
```

SOCK_CLOSE

Type : Close a socket

Syntax : result% = SOCK_CLOSE(<socket%>)

See Also : SOCK_INIT

Notes : None

Result : Close the socket defined by **socket%**. If there was no error, then **TRUE** is returned. Any other value denotes an error.

Example :

```
// This example closes the socket number 1024
A = SOCK_CLOSE(1024)
```

SOCK_GETIP

Type : Convert web or IP address to IPV4 value

Syntax : `result% = SOCK_GETIP(<address$>)`

See Also : SOCK_GETIP$

Notes :

- The socket system needs to be initialised before this command can be used
- This command returns -1 if there was an error
- An IP address of 255.255.255.255 is used to allow broadcasting to all computers in the domain range

Result : Tries to convert the given address (which could be a web address, IP address or computer name), held in **address$** into an IP address.

Example :

```
// This example converts the IP address to a 32-bit IP address number
A% = SOCK_GETIP("127.0.0.1")
```

SOCK_GETIP$

Type : Convert an IPV4 address to a readable string

Syntax : `result$ = SOCK_GETIP$(<address%)`

See Also : SOCK_GETIP

Notes :

- The socket system does not need to be initialised before this command can be used

Result : Tries to convert an binary IP address to a user-readable string. The IP address to be converted is stored in **address%**.

Example :

```
// This example converts an IP address value into a readable string, with the format of
// xxxx.xxxx.xxxx.xxxx
A$ = SOCK_GETIP$(2130706433)
```

SOCK_GETREMOTEIP

Type : Get IP address of connected remote computer

Syntax : ip% = SOCK_GETREMOTEIP(<socket%>)

See Also : SOCK_GETREMOTEPORT

Notes : None

Result : Returns the IP Address of the connected remote computer. For UDP connections, this value is set with each SOCK_RECV call

Example :

```
// Get IP address of the computer that last sent a message
addr%=SOCK_GETREMOTEIP(socket%)
```

SOCK_GETREMOTEPORT

Type : Get port of connected remote computer

Syntax : port% = SOCK_GETREMOTEIP(<socket%>)

See Also : SOCK_GETREMOTEPORT

Notes : None

Result : Returns the port of the connected remote computer. For UDP connections, this value is set with each SOCK_RECV call

Example :

```
// Get port number of the computer that last sent a message
port%=SOCK_GETREMOTEIP(socket%)
```

SOCK_INIT

Type : Enable the socket set of commands

Syntax : `result% = SOCK_INIT()`

See Also : SOCK_SHUTDOWN, NETGETLASTERROR$

Notes : None

Result : Initialising the socket networking system. This command returns **TRUE** if there was no problems, and **FALSE** if there was.

Example :

```
// This example initialises the Socket system
A = SOCK_INIT()
```

SOCK_RECV

Type : Received a TCP or UDP message

Syntax : `result% = SOCK_RECV(<socket>, <message$>, <length%>)`

See Also : NETGETLASTERROR$

Notes : None

Result : Receive a message from the socket defined by **socket%**.

message$ holds the received message or an empty string if there was an error.

length% defines the total size of the message - the amount received could be less than this, and is sent as the return value in addition to other values :

Return value of SOCK_RECV	Meaning
-2	A socket is not ready to be read. Try reading from the socket later on.
-1	There was an error (use NETGETLASTERROR$) to see what the error is
0	If receiving a TCP message, the sending machine has been disconnected, and this should be disconnected
> 0	The number of successfully received bytes

Example :

```
// This example receives any messages into MESSAGE$, with a maximum length of 1024 bytes.
// The result is stored in A
A = SOCK_RECV(SOCKET, MESSAGE$, 1024)
```

SOCK_SHUTDOWN

Type : Shuts down the socket system

Syntax : SOCK_SHUTDOWN

See Also : SOCK_INIT

Notes : None

Result : Shuts down the socket system

Example :

```
// This example shuts down the socket system
SOCK_SHUTDOWN
```

SOCK_TCPACCEPT

Type : Return a socket if a computer has successfully connected to a port

Syntax : sockRet% = SOCK_TCPACCEPT(sock, ip%)

See Also : NETGETLASTERROR$, SOCK_TCPCONNECT, SOCK_TCPLISTEN

Notes : None

Result : This command retrieves a valid socket to a computer that has successfully used SOCK_CONNECT after responding to SOCK_TCPLIST

The IP address of the successful computer is stored in **ip%**.

The servers socket ID is stored in **sock%**.

Example :

```
// This example returns a socket ID and stores the IP address of the connecting computer in the
// variable  IP
A% = SOCK_TCPACCEPT(SOCK, IP)
```

SOCK_TCPCONNECT

Type : Connect to a TCP server

Syntax : `sock% = SOCK_TCPCONNECT(<server$>, <port%>)`

See Also : `NETGETLASTERROR$, SOCK_TCPACCEPT, SOCK_TCPLISTEN`

Notes : None

Result : Try to connect to the server defined by **server$**, using the port number defined by **port%**

This command returns a socket number if there was no error, or -1 if there was.

server$ can be either an IP address or name

Example :

```
// This example connects to the server at 192.168.1.1 using port 1024
A = SOCK_TCPCONNECT("192.168.1.1", 1024)
```

SOCK_TCPLISTEN

Type : Listens for a TCP message

Syntax : `sock% = SOCK_TCPLISTEN(<port%>)`

See Also : `NETGETLASTERROR$, SOCK_TCPCONNECT, SOCK_TCPACCEPT`

Notes : None

Result : Listens for TCP messages on the port specified by **port%**. The return value is either a socket number if there was no problems, or -1 if there was.

Example :

```
// This example listens for a TCP message on port 80
SOCK = SOCK_TCPLISTEN(80)
```

SOCK_TCPSEND

Type : Send a message to a TCP connected socket

Syntax : `result% = SOCK_TCPSEND(<sock%>, <msg$>)`

See Also : `NETGETLASTERROR$, SOCK_TCPCONNECT, SOCK_TCPACCEPT`

Notes : None

Result : This command sends a message (stored in **msg$**) to a connected socket, which is stored in **sock%**.

If there was an error, then -1 is returned, otherwise the number of characters sent is returned.

Example :

```
// This example sends a message to the socket stored in SOCK
A = SOCK_TCPSEND(SOCK, "THIS IS A MESSAGE")
```

SOCK_UDPOPEN

Type : Open a socket for UDP transmission

Syntax : `result% = SOCK_UDPOPEN(<port%>)`

See Also : `NETGETLASTERROR$`

Notes :

- Sockets in GLBasic have SO_BROADCAST flag set - this means that all messages sent are also received by the sender
- The value for result will start at 0 (if there was no error)

Result : This command opens a UDP socket that enables receiving and sending of UDP messages on the port defined by **port%**

The command returns a valid socket if there was no problem, or -1 if there was an error.

Example :

```
// This example opens a UDP socket on port 32767
A = SOCK_UDPOPEN(32767)
```

SOCK_UDPSEND

Type : Send a message using an UDP socket

Syntax : `result% = SOCK_UDPSEND(<socket%>, <message$>, <destIP%>, <destPort%>)`

See Also : NETGETLASTERROR$, SOCK_RECV

Notes :

- From V7.170, CHR$(0) send be sent in message$

Result : This command sends a UDP message to the IP address defined by **destIP%**. The recipient port number is defined by **destPort%**.

The message you want to send is defined by **message$**, and the open socket to be used is defined by socket%.

The return value indicates the number of bytes sent. If this is < 0, then an error occurred.

Example :

```
// This example sends a message to the IP address held in DESPIP, and uses port 32767
A = SOCK_UDPSEND(SOCK, "THIS IS A MESSAGE", DESPIP%, 32767)
```

SORTARRAY

Type : Sorts an array

Syntax : SORTARRAY array[], [ADDRESSOF(<*compare*>) | 0]

See Also : ADDRESSOF

Notes :

- The **compare** function should only be defined as returning a floating-point value. If it is defined as returning an integer, your program will crash
- The default sorting routine (when 0 is used), will compare (if a type is used), the first member of the array's type.

Result : This function allows the sorting of arrays, or arrays of a TYPE, defined by array, the default is to sort data in ascending order.

If the second parameter uses ADDRESSOF, then ***compare*** must be a name of a function. If the second parameter is a value of 0, then the default sorting routine is used.

Example :

```
// An example compare routine
FUNCTION COMPARE:a as tType,b as tType
   IF a.value<b.value THEN RETURN -1
   IF a.value>b.value THEN RETURN 1
   RETURN 0
ENDFUNCTION

// This example uses the user-defined function called 'compare' to help sort the items in the array
// called 'array'
SORTARRAY ARRAY[], ADDRESSOF(COMPARE)

// This example sorts an array using the standard sorting routine
SORTARRAY ARRAY[], 0
```

SOUNDPLAYING

Type : Determines whether a sound is being played

Syntax : `result% = SOUNDPLAYING(<channel%>)`

See Also : LOADSOUND, PLAYSOUND

Notes : None

Result : This command determines whether the sound attached to the sound channel defined by **channel%** is being played or not.

It returns **TRUE** if a sound is being played, and **FALSE** if it is not.

Example :

```
// This example returns TRUE if the sound from channel 0 is being played
RESULT% = SOUNDPLAYING(0)
```

SPLITSTR

Type : Splits the contents of a string into separate sections

Syntax : `result% = SPLITSTR(<text$>, <array$[]>, <seperator$> [, <dropEmpty%> = TRUE])`

See Also : MID$, LEFT$, RIGHT$

Notes :

- If array$[] is a multi-dimensional variable, then the result(s) are only put in the first index.

Result : Splits the contents held in **text$** into separate sections, with each section being store in a new index of the array **array$**. Each section is delimited by the contents of **seperator$**

The optional parameter **dropEmpty%** denotes whether any empty parts between any two separators are ignored (**dropEmpty%** = **TRUE**) or whether they should be added to the array list (held in array$). Using a **dropEmpty%** value of **TRUE** allows your program to emulate the PHP *split* commands

This command returns the number of sections the text has been split into.

Example :

```
// This stores all text that is delimited by a "," into the array called ARRAY$
// It would contain :
// THIS
// IS
// A
// TEST
// And A would hold 4
A = SPLITSTR("THIS, IS, A, TEST", ARRAY$[], ",")
```

SPRCOLL

Type : Perform collision detection between two sprites

Syntax : `result% = SPRCOLL(<index1%>, <x1%>, <y1%>, <index2%>, <x22%>, <y2%>)`

See Also : ANIMCOLL

Notes :

- This commands calls BOXCOLL before pixel based collision is performed, in order to speed up collision detection

Result : Performs per-pixel collision detection between two sprites (defined by **index1%** and **index2%**).

The horizontal position is defined by **x1%** and **x2%**, and the vertical is defined by **y1%** and **y2%**.

This command returns **TRUE** if a collision has been detected, otherwise it returns **FALSE**.

Example :

```
// This example checks to see if sprite 0 at coordinates 100 by 100 collides with sprite 1 at
// coordinates 110 by 120
A = SPRCOLL(0, 100, 100, 1, 110, 120)
```

SPRITE2MEM

Type : Put graphic data of a sprite into an array

Syntax : `result% = SPRITE2MEM(<array%[]>,<sprite%>)`

See Also : `MEM2SPRITE`

Notes : None

Result : This command puts a sprites graphic data into the array defined by **array%[]**.

sprite% defines which sprite number to retrieve the data from

Example :

```
// This retrieves the graphical data from sprite 0
result%=sprite2mem(array%[],0)
```

SQR

Type : Returns the square root of a value

Syntax : `result = SQR(<value>)`

See Also : `ASIN`, `ACOS`, `ATAN`, `SIN`, `COS`, `TAN`

Notes : None

Result : Returns the square root held in value, and stores it into result.

Example :

```
// Sets A to be 9.486832981
A = SQR(90.0)
```

STARTDATA

Type : Start a data block

Syntax : STARTDATA <label:>

See Also : ENDDATA, DATA

Notes : None

Result : Start a data block, giving it the label defined by **label**. This name can be used by RESTORE to define a data block to be read

Example :

```
// Define a data block, giving it the name of “letters”
STARTDATA letters:
   DATA “A”,”B”,”C
ENDDATA
```

STARTPOLY

Type : Allow the drawing of polygons

Syntax : STARTPOLY spriteID% [,*bMode%=0*]

See Also : POLYVECTOR, ENDPOLY

Notes :

- No graphic access using other commands like DRAWSPRITE, DRAWLINE etc. should be performed during a STARTPOLY and ENDPOLY block

Result : This command allows the drawing of polygons, and places the result in the sprite ID defined by **spriteID%**. If **bMode%** is present, then it details how the **spriteID%** or triangles will be drawn :

Value of **bMode%**	Result
-1	No sprite is to be used
0	Triangle fan
1	Single triangles
2	Triangle strip

Example :

```
// This example creates a poly-vector starting at 1,1 and with a width and height of 8 pixels. It
// has a purple colour
STARTPOLY 500
   POLYVECTOR 1,1, 8, 8,RGB(255,0,255)
ENDPOLY

// The following more complex example creates a rectangle with gradient colouring from each of the
// four corners
FUNCTION box:x%,y%,width%,height%,c1%,c2%,c3%,c4%
   STARTPOLY 500
      POLYVECTOR width%,y%,0,0,c1%
      POLYVECTOR x%,y%,0,0,c2%
      POLYVECTOR x%,height%,0,0,c3%
      POLYVECTOR width%,height%,0,0,c4%
   ENDPOLY
ENDFUNCTION
```

STATIC

Type : Defines a variable in a function which will exist for the life of the program

Syntax : STATIC <var>

See Also : LOCAL, GLOBAL

Notes : None

Result : Defines the variable **var**, which exists even when the function has finished. When the function is re-entered, the variable will have the previously stored value

Example :

```
// The standard definition is like the following example
STATIC one

// This example defines the static variable 'two', which will exist when the function finishes
FUNCTION test:
  STATIC two
ENDFUNCTION
```

STDCOLOR

Type : Sets the background and foreground colour for the next STDOUT command

Syntax : `STDCOLOR` *`[fore][, back]`*

See Also : `STDOUT`

Notes : None

Result : The next `STDOUT` command uses the colour values set by this command.

The foreground colour is set by fore, and the background colour is set by back. By default, the colours used (if none are give) are 15 and 1

Each value should be between 0 and 15, and correspond to the following colours :

fore or **back** colour	Colour
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Purple
6	Yellow
7	Light Grey
8	Grey
9	Light Blue
10	Light Green
11	Light Cyan
12	Light Red
13	Light Purple
14	Light Yellow
15	White

Example :

```
// Set foreground colour to grey and the background colour to white
STDCOLOR 8, 15
```

STDERR

Type : Sends text to the standard error stream

Syntax : `STDERR <text$>`

See Also : `STDCOLOR`, `STDOUT`

Notes : None

Result : Sends the text held in **text$** to the standard error stream

Example :

```
// Sends text to the standard error stream.  This could be a console window
STDERR "AN ERROR"
```

STDIN$

Type : Reads text in from the standard input stream, which is usually a console window

Syntax : `result$ = STDIN$()`

See Also : `STDOUT`

Notes : None

Result : Reads in text from the standard input stream (which is usually a console window), and stores the result in **result$**

The input is terminated when the user enters a NEWLINE command (presses *RETURN*), which is removed from the result.

The maximum length of the entered text can be 4095 characters

Example :

```
// Waits until user presses RETURN and stores what has been typed into the variable A$
A$ = STDIN$()
```

STDOUT

Type : Sends text to the standard output stream, which is usually a console window

Syntax : STDOUT <text$>

See Also : STDCOLOR, STDERR

Notes : None

Result : Sends the text held in **text$** to the standard output stream. Usually this is a console window .

Example :

```
// Sends the text to the console window
STDOUT "SENT TO CONSOLE"
```

STEP

Type : Increases FOR counter by a defined amount

Syntax : STEP <var>

See Also : FOR, NEXT

Notes :

- If the step value is to be 1, then the STEP command does not need to be given.

Result : Increases the FOR counter by the step value of **var**. This can be either positive or negative. It can only be used with the FOR statement

Example :

```
// This example increases A by 2
FOR A = 1 TO 16 STEP 2
NEXT

// This example decreases A by 3
FOR A = 100 TO 0 STEP -3
NEXT
```

STOPMUSIC

Type : Stops music being played (but not sounds)

Syntax : STOPMUSIC

See Also : PLAYMUSIC, HUSH

Notes : None

Result : Any music being played (using LOADMUSIC) will be stopped

Example :

```
// This example stops all music being played
STOPMUSIC
```

STOPSOUND

Type : Stops a sound being played

Syntax : STOPSOUND <channel%>

See Also : PLAYSOUND

Notes : None

Result : Any sounds being played by the sound channel **channel%** will be stopped

Example :

```
// This example stops the sound being played by channel 1
STOPSOUND 1
```

STRETCHANIM

Type : Draws and resizes an animated sprite

Syntax : `STRETCHANIM <index%>, <tile%>, <x%>, <y%>, <width%>, <height%>`

See Also : `STRETCHSPRITE`

Notes : None

Result : Resizes a sprite from sprite index index, with a tile index of **tile%**.

Horizontal positioning is set by **x**, and vertical positioning is defined by **y**.

The sprite with be drawn with the new size specified by the width value of **width%**, and the height value of **height%**.

Example :

```
// This example resizes tile 0 in sprite 1 to be 128 bytes by 128 bytes and positions it at
// coordinates 100 by 100
STRETCHANIM 1, 0, 100, 100, 128, 128
```

STRETCHSPRITE

Type : Draws and resizes a sprite

Syntax : STRETCHSPRITE <index%>, <x%>, <y%>, <width%>, <height%>

See Also : DRAWSPRITE, ROTOSPRITE, ROTOZOOMSPRITE, STRETCHANIM, ZOOMSPRITE

Notes : None

Result : Resizes a sprite from sprite index **index%**.

Horizontal positioning is set by **x%**, and vertical positioning is defined by **y%**.

The sprite with be drawn with the new size specified by the width value of **width%**, and the height value of **height%**.

Example :

```
// This example resizes sprite 1 to be 64 pixels by 64, and positions it at coordinates 100 by 100
STRETCHSPRITE 1, 100, 100, 64, 64
```

SUB

Type : Defines a subroutine that has no arguments and returns no values

Syntax : SUB <name:>

See Also : ENDSUB, CALLBYNAME, GOSUB

Notes : None

Result : Defines a subroutine called name, which GOSUB or CALLBYNAME will use to call the routine

Example :

```
// This example creates a subroutine called 'name' that can be called by either using
// CALLBYNAME or GOSUB
SUB name:
ENDSUB
```

SWAP

Type : Swap the value of two variables

Syntax : SWAP <value1> | <value1%> | <value1$>,
<value2> | <value2%> | <value2$>

See Also : None

Notes :

- The variable types for **value1** and **value2** don't have to be the same

Result : This swaps the contents of **value1** and **value2**, so that the value held originally by **value2** is in **value1** and vice-versa.

Example :

```
A%=1
B%=2
SWAP A%, B% // After the command is executed, a% now holds 2 and b% holds 1

C$="Cat"
D$="Dog"
SWAP C$,D$ // C$ holds Dog and D$ holds Cat

A%=7
B$="24"
SWAP A%,B$ // A% holds 24 and B$ holds 7
```

SYSTEMPOINTER

Type : Enables or disables the system mouse pointer

Syntax : SYSTEMPOINTER <mode%>

See Also : None

Notes : None

Result : Enables the mouse pointer if **mode%** = **TRUE**, and disables it if **mode%** = **FALSE**

Example :

```
// This example displays the system mouse pointer
SYSTEMPOINTER TRUE
```

TAN

Type : Returns a the tangent of a value

Syntax : `result = TAN(value)`

See Also : `ASIN`, `ACOS`, `ATAN`, `SIN`, `COS`

Notes : None

Result : Returns the tangent held in **value**.

Example :

```
// Sets A to be 0.3521555903
A = TAN(19.4)
```

THEN

Type : Optional item which defines what action to be taken if the `IF` statement evaluates to **TRUE**

Syntax : `THEN <result>`

See Also : `IF`

Notes :

- `ELSE` and `ELSEIF` cannot be used if `THEN` is also used.

Result : Only needed if a single-line `IF` statement is to be used, this statement causes **result** to be actioned if the proceeding `IF` evaluates to **TRUE**

Example :

```
// This example sends the text “IT IS TRUE” to the debug window if A = 1
IF A = 1 THEN DEBUG “IT IS TRUE!”
```

THROW

Type : Generate an exception error

Syntax : THROW <error$>

See Also : TRY, CATCH, FINALLY

Notes :

- This command should only be used with a TRY/FINALLY block. Used on its own will crash your program

Result : Calls the CATCH section of the last-used TRY/FINALLY block, passing it the text held in **error$**

Example :

```
// The general example for THROW is :
THROW "Message"

// The general usage for THROW is as follows, which calls the CATCH section if A% is 0 - which will
display the sent message to the debug window
TRY
  // Code to be checked
  IF A%=0
     THROW "A% is 0!"
  ENDIF
CATCH ERROR$
  // ERROR$ will contain "A% is 0!" The full error message displayed will be 'The error is A% is 0!'
  DEBUG "The error is : "+ ERROR$
FINALLY
```

TO

Type : Introduces terminating condition in FOR/NEXT loop

Syntax : TO <value>

See Also : FOR, NEXT

Notes : None

Result : Defines the terminating condition, held in **value**, to be used by the proceeding FOR statement

Example :

```
// This is the basic usage.  Also see the FOR example
TO 16
```

TRIM$

Type : Trims text from the beginning and end of text

Syntax : result$ = TRIM$(<text$> [, *<remove$="\t\r\n\v\f">*])

See Also : LTRIM$, RTRIM$

Notes : None

Result : This removes all text held in **remove$** from the text in **text$**. Text is removed from the left and right-hand side of the text.

Example :

```
// This example removes all “a”’s from the start and end of the text “abcdefa”
DEBUG TRIM$(“abcdefa”,”a”)
```

TRUE

Type : Holds a value of 1

Syntax : <var> = TRUE

See Also : FALSE

Notes : None

Result : Used to give a variable a specific value or to determine whether something is valid/correct etc.

Example :

```
a%=TRUE
```

TRY

Type : Initiate the ability to catch errors (exceptions) within a block of statements

Syntax : TRY

See Also : CATCH, FINALLY, THROW

Notes :

- The CATCH statement must be included, otherwise a compiler error will be generated

Result : Initiates exception handling

Example :

```
// The general usage for TRY is :
TRY

// The general usage for TRY is:
TRY
  // Code to be checked
CATCH
  // Process any errors here
FINALLY
```

TYPE

Type : Allows creation of groups of variables and basic object oriented routines (from V8 onwards)

Syntax : TYPE <name>

See Also : ENDTYPE, self

Notes : None

Result : Starts the ability to create a structure, giving it the name of **name**. With version 8, TYPE allows functions to be included, allowing basic object-oriented programming.

Example :

```
// The command layout is as follows :
TYPE name
  ...
ENDTYPE

// This example defines a TYPE with the name of Player, with the variables in it
TYPE player
   X
   Y
   NAME$
ENDTYPE

// This example shows how to use functions in TYPE's:
TYPE Ta
  FUNCTION test%:
  ENDFUNCTION
  FUNCTION test2%:
    self.test() // Call test that is in the type Ta
  ENDFUNCTION
ENDTYPE
```

UCASE$

Type : Converts any characters to their upper case equivalent

Syntax : `result$ = UCASE$(<text$>)`

See Also : `LCASE$`

Notes : None

Result : Converts all lower case characters (a to z) held in **text$**, into their upper case equivalent (A to Z). All other characters are left are they are.

Example :

```
// This example converts the text to lower case letters and stores it in A$
// The variable would be holding 'THIS IS A TEST'
LOCAL A$
A$ = UCASE$("this is a test")
PRINT A$, 10, 10
SHOWSCREEN
KEYWAIT
```

UNTIL

Type : Evaluate condition to see if the `REPEAT`/`UNTIL` block should exit

Syntax : `UNTIL <condition> = TRUE`

See Also : `REPEAT`

Notes : None

Result : `REPEAT`/`UNTIL` will loop until the specified **condition** evaluates to **TRUE**

Example :

```
// General usage
REPEAT
UNTIL a condition = TRUE

// Loop until A > 10
REPEAT
 ...
UNTIL A>10
```

URLDECODE$

Type : Decode any %## encoding to original string text

Syntax : `result$ = URLDECODE(<code$>)`

See Also : URLENCODE$

Notes : None

Result : This command converts any encodings beginning with a % and followed by a hexadecimal number to be converted into the original string characters.

For example %20 is the hexadecimal value of 32, which is a space.

code$ is the text that you want to convert back to the original text. This command is mainly used for converting URL strings to the actual text.

Example :

```
LOCAL enc$ = URLENCODE$("?mail=me@privacy.net&text=some/special\nchars")
STDOUT enc$ + "\n"
STDOUT URLDECODE$(enc$) + "\n"
```

URLENDCODE$

Type : Encode any escape codes into #xx format

Syntax : `result$ = URLENCODE(<code$>)`

See Also : `URLENCODE$`

Notes : None

Result : This command converts any text held in code$ that is an escape code to a format of #xx, where xx is the hexadecimal value of the escape code.

For example, “A Test” would be converted to A%20Test”

This command is mainly used with URL access, to allow characters that would otherwise be disallowed, to be sent to a website.

Example :

```
LOCAL enc$ = URLENCODE$("?mail=me@privacy.net&text=some/special\nchars")
STDOUT enc$ + "\n"
STDOUT URLDECODE$(enc$) + "\n"
```

USEASBMP

Type : Use the current background buffer as a background image

Syntax : `USEASBMP`

See Also : `SHOWSCREEN`

Notes : None

Result : This command allows the use of the contents of the background buffer as the background image at sequent `SHOWSCREEN` calls

Example :

```
// This uses the current background buffer as a background image
USEASBMP
```

USESCREEN

Type : Performs all graphical output on a defined virtual screen

Syntax : USESCREEN <screen%>

See Also : CREATESCREEN

Notes : None

Result : This command allows all graphical output to sent to a virtual screen, defined by **screen**.
If **screen** = -1, then all output reverts back to the back buffer

Example :

```
// This example uses screen 0 for all output
USESCREEN 0
```

VIEWPORT

Type : Defines area in which graphics are plotted and displayed

Syntax : `VIEWPORT <x%>, <y%>, <width%>, <height%>`

See Also : `GETVIEWPORT`

Notes :

- When using this command in a 3D environment, this command MUST come before `X_MAKE3D`, otherwise all objects will either not be displayed or disappear after the first screen refresh
- All x and y positions start at coordinates (0,0) to (width%-1,height%-1)
- Currently using this command will set `SMOOTHSHADING` to **`FALSE`**

Result : This command defines the area at which graphics can be plotted and displayed.
The starting x and y coordinates are defined by **x** and **y**, whilst **width** and **height** define the width and height of the viewport.

Anything plotted outside these coordinates is not displayed.

3D graphics are scaled to the size of the viewport

Setting **width%** or **height%** to be less than or equal to 0 will reset the viewport to the width and height of the current screen size

Example :

```
// This example creates a view port from coordinates 0,0 to 640,480
VIEWPORT 0,0, 640, 480

// This example resets the viewport to the full graphics screen width and height
VIEWPORT 0,0,0,0
```

WEND

Type : Denotes the end of a `WHILE` loop

Syntax : `WEND`

See Also : `WHILE, REPEAT, UNTIL`

Notes : None

Result : Denotes the end of a `WHILE` loop. Without this, `WHILE` loops could run into each other causing compiler problems

Example :

```
WHILE
    ...
    // Terminate the WHILE loop
WEND
```

WHILE

Type : Repeat a set of commands whilst a condition evaluates to **TRUE**

Syntax : `WHILE <evaluation> = TRUE`

See Also : `WEND, REPEAT, UNTIL`

Notes : None

Result : Evaluates **evaluation**, and if **TRUE** starts the execution of the code between the `WHILE` and `WEND` commands.

Example :

```
// A standard use for WHILE.  This example will continue executing code between WHILE and WEND until
// A does not equal 5
WHILE A = 5
WEND
```

WRITEBYTE

Type : Writes a signed-byte to an opened file

Syntax : `WRITEBYTE <handle%>, <value%>`

See Also : `WRITEUBYTE`

Notes : None

Result : Writes value (as a value between -127 and +127) to the opened file held in **handle%**.

The value written is held by **write%**.

Example :

```
// This example writes a signed byte to a file
WRITEBYTE 1, W%
```

WRITEIEEE

Type : Writes a double-precision floating-point value to an opened file

Syntax : `WRITEIEEE <handle%>, <write>`

See Also : `WRITESHORTIEEE`

Notes : None

Result : Writes a double-precision (8 byte) floating-point value to the opened file held in **handle%**. The value to be written is stored in **write**.

Example :

```
// This example writes a double float to a file
WRITEIEEE 1, W
```

WRITELINE

Type : Writes data to a file

Syntax : `WRITELINE <handle%>, <write$>`

See Also : `READLINE`

Notes : None

Result : Writes data, held in **write$**, to the opened file held in **handle%**. A line terminator ("\r\n") set of characters are added to the data written.

Example :

```
// This example writes DATA to a file
WRITELINE 1, "DATA"
```

WRITELONG

Type : Writes a signed-long to an opened file

Syntax : `WRITELONG <handle%>, <write%>`

See Also : `WRITEULONG`

Notes : None

Result : Writes a signed-long value (as a value between -2,147,483,648 and +2,147,483,647) to the opened file held in **handle%**. The value to be written is held in **write%**.

Example :

```
// This example writes a signed long to a file
WRITELONG 1, W%
```

WRITESHORTIEEE

Type : Writes a single-precision floating-point value to an opened file

Syntax : `WRITESHORTIEEE <handle%>, <write>`

See Also : `WRITEIEEE`

Notes : None

Result : Writes a single-precision (4 byte) floating-point value to the opened file held in **handle%**. The value to be written in stored in **write**.

Example :

```
// This example writes a float to a file
WRITESHORTIEEE 1, W
```

WRITESTR

Type : Writes a given number of characters to an opened file

Syntax : `WRITESTR <handle%>, <write$>, count%`

See Also : `READSTR`

Notes : None

Result : Writes a given number of characters to the opened file held in **handle%**, from the data held in **write$**. The number of characters is held in **count%**.

Example :

```
// This example writes 5 characters held in text$ to a file
WRITESTR 1, text$, 5
```

WRITEUBYTE

Type : Writes an unsigned-byte to an opened file

Syntax : `WRITEUBYTE <handle%>, <write%>`

See Also : `WRITEBYTE`

Notes : None

Result : Writes a byte (as a value between 0 and 255) to the opened file held in **handle%**.

The value to be written is held in **write%**.

Example :

```
// This example writes an unsigned byte to a file
WRITEUBYTE 1, W%
```

WRITEULONG

Type : Writes an unsigned long to an opened file

Syntax : `WRITEULONG <handle%>, <write%>`

See Also : `WRITELONG`

Notes : None

Result : Writes an unsigned long (as a value between 0 and +4,294,967,295) to the opened file held in **handle%**. The value to be written is stored in **write%**

Example :

```
// This example writes an unsigned long to a file
WRITEULONG 1, W%
```

WRITEUWORD

Type : Writes an unsigned word to an opened file

Syntax : `WRITEUWORD <handle%>, <write%>`

See Also : `WRITEWORD`

Notes : None

Result : Writes an unsigned word (as a value between 0 and +65535) to the opened file held in **handle %**. The value to be written is held in **write%**.

Example :

```
// This example writes an unsigned word to a file
WRITEUWORD 1, W%
```

WRITEWORD

Type : Writes a signed word to an opened file

Syntax : `WRITEWORD <handle%>, <write%>`

See Also : `WRITEUWORD`

Notes : None

Result : Writes a signed-word (as a value between -32767 and +32767) to the opened file held in **handle%**.

The value to be written is stored in **write%**.

Example :

```
// This example writes a signed word to a file
WRITEWORD 1, W%
```

X_AMBIENT_LT

Type : Enables ambient lighting

Syntax : X_AMBIENT_LT <light%>, <colour%>

See Also : X_SPOT_LT

Notes : None

Result : Enables ambient lighting. Which light you want to use is defined by **light%**, and should be between 0 and 7.

The colour to be used is defined by **colour%**.

Example :

```
// This example sets the ambient light 0 to yellow
X_AMBIENT_LT 0, RGB(255,0,255)
```

X_AUTONORMALS

Type : Choose mode for lighting at the next user-created object

Syntax : X_AUTONORMALS <mode%>

See Also : None

Notes :

- Will only work with objects created programmatically and not those loaded with X_LOADOBJ

Result : Choose which mode lighting will use on normal calculations when the next object is created.

mode% can be :

Value of **mode%**	Description
0	None
1	Flat – Cubes have real edges
2	Surface normals - all triangles that share a common point are assumed to belong to the same surface. One normal vertex for this surface will be used for each shared point, this edges will be smoothed and blended

Example :

```
// This example sets lighting so that cubes have real edges
X_AUTONORMALS 1
```

X_CAMERA

Type : Create a camera

Syntax : X_CAMERA <x>, <y>, <z>, <px>, <py>, <pz>

See Also : X_CAMERA_UP

Notes :

- At least one camera is needed to view 3D objects

Result : Create a camera at the positions defined by **x**, **y** and **z**. The camera will then point in the direction defined by **px**, **py** and **pz**.

Example :

```
// This example sets the camera to position 0, 0, 0 and points to 100, 0, 0
X_CAMERA 0.0, 0.0, 0.0, 100. 0, 0.0, 0.0
```

X_CAMERAUP

Type : Set camera's up direction

Syntax : X_CAMERA_UP <x>, <y>, <z>

See Also : X_CAMERA

Notes : None

Result : Set the camera's up direction. This needs to be used before **X_CAMERA** The **x**, **y** and **z** values correspond to the x, y and z directions, and take either 1.0 or 0.0 as their values

Example :

```
// This example  sets the camera up position so that the X direction is regarded as up
X_CAMERA_UP 1.0, 0.0, 0.0
```

X_CLEAR_Z

Type : Clear the depth buffer

Syntax : X_CLEAR_Z

See Also : None

Notes : None

Result : Clears the depth buffer. This allows for drawing over this like sky boxes.

Example :

```
// This example clears the depth buffer
X_CLEAR_Z
```

X_COLLISION

Type : Performs a collision check between two objects

Syntax :
```
result% = X_COLLISION(<index%>, <frame>, <radius>,
                      <x>, <y>, <z>)
```

See Also : X_COLLISIONAABB

Notes : None

Result : Performs a spherical object collision check, returning the ID number of the object that has collided with the object defined by **index%**.

frame denotes the animation frame of the object index, radius is the size of the sphere in which the collision check will be performed.

x, **y** and **z** denote the centre of the sphere.

Example :

```
// This example checks to see if  there is a collision between 3D object 0 (with animation frame
// 0) .  The radius to check is 32 units and the position to start checking is at 100, 100, 100
A = X_COLLISION(0, 0, 32.0, 100.0, 100.0, 100.0)
```

X_COLLISIONAABB

Type : Performs a bounding box collision check between two objects

Syntax : `result% = X_COLLISIONAABB(<index%>, <frame>, <x>, <y>, <z>, <width>, <height>, <depth>)`

See Also : X_COLLISION

Notes : None

Result : Performs a axis aligned bounding box object collision check, returning the ID number of the object that has collided with the object defined by **index%**.

frame denotes the animation frame of the object index.

x, **y** and **z** denote the centre of the box, whilst width, height and depth denote the size of the bounding box.

Example :

```
// This example checks for a collision from 3D object 0 and animation frame 0, and at position 100,
// 100, 100.  The size to check for is 32 units high, 32 units wide and 32 units in depth
A = COLLISIONAABB(0, 0, 100.0, 100.0, 100.0, 32.0, 32.0, 32.0)
```

X_COLLISIONRAY

Type : Performs a ray intersection collision check

Syntax : `result = X_COLLISIONRAY(<index%>, <frame>, <x>, <y>, <z>, <dirX>, <dirY>, <dirZ>)`

See Also : `X_COLLISION, X_COLLISIONAABB`

Notes : None

Result : Performs a ray intersection collision check.

index% is the object ID to check, **frame** is the animation from of the object, **x**, **y** and **z** is the 3D position of the object.

dirX, **dirY** and **dirZ** is the direction that the ray uses.

The value of result will be one of the following :

Return result of `X_COLLISONRAY`	Meaning
< 0	No collision in positive direction found
0	No collisions found
> 0	Distance to collision in the direction of **dirX**, **dirY**, **dirZ**

Example :

```
// This example checks to if the 3D object index 0,using frame 0, at position 100, 100, 100 collides
// with any object  0 units in front, 16 units up and 0 units along
A = X_COLLISIONRAY(0, 0, 100.0, 100.0, 100.0, 0.0, 16.0, 0.0)
```

X_COLLISIONSPHEREBOX

Type : Performs a cuboid collision against a sphere

Syntax : `result% = X_COLLISIONSPHEREBOX(<cx>, <cy>, <cz>, <radius>, <bx>, <by>, <bz>)`

See Also : X_COLLISION, X_COLLISIONAABB, X_COLLISIONRAY

Notes : None

Result : Performs a cuboid collision (defined by position coordinates defined by **bx**, **by** and **bz**), after transformation with the current matrix (X_MOVEMENT, X_ROTATION or X_MULTIMATRIX) against a sphere defined by **cx**, **cy** and **cz** with a radius defined by **radius**.

This command returns ID number of object that has collided at the coordinates.

Example :

```
// This example checks to see if there is a collision between an object at 0, 0, 0 and with a radius
// of 32, against an object at 16, 16, 16
A = X_COLLISIONSPHEREBOX(0.0, 0.0, 0.0, 32.0, 16.0, 16.0, 16.0)
```

X_CULLMODE

Type : Determine how a 3D object is displayed

Syntax : X_CULLMODE <mode%>

See Also : None

Notes : None

Result : Determine how a 3D object is displayed, based on the value of **mode%** :

Value of **mode%**	What is drawn :
-1	Only draw back sides
0	Front and back sides are drawn
1	Only front sides are drawn

Example :

```
// This example sets culling to draw front and back side
X_CULLMODE 0
```

X_DOT

Type : Draws a 3D dot

Syntax : X_DOT <x>, <y>, <z>, <size>, <colour%>

See Also : X_LINE

Notes : None

Result : Draws a 3D dot at positions **x**, **y** and **z**, with a size defined by **size**. The colour to be used is defined by **colour%**.

Example :

```
// This example puts a 3D dot at position 100, 100, 100 with a size of 2 units and a colour of red
X_DOT 100.0, 100.0, 100.0, 2.0, A1RGB(255, 0, 0)
```

X_DRAWANIM

Type : Draw animation frame of a 3D object

Syntax : `X_DRAWANIM <index%>, <from>, <to>, <interpol>, <fullAnim%>`

See Also : `X_DRAWOBJ`

Notes : None

Result : Draws an interpolated animation from the object index defined by **index%**. **to** and **from** specify the start and end animation frames to interpolate between.

interpolate defines the current state of interpolation between the starting and ending frames and should be a value between 0.0 and 1.0

fullAnim% defines how the interpolation is done. If this parameter is set to **FALSE** then interpolation will be performed with from and to values. If **fullAnim%** is set to **TRUE**, then interpolation is performed for all frames between from and to.

Example :

```
// This example draws an animation from of 3D object 0 - the frame index is between 0 and 16
X_DRAWANIM 0, 0, 16, 0.1, TRUE
```

X_DRAWAXES

Type : Draws debugging axes

Syntax : `X_DRAWAXES <x>, <y>, <z>`

See Also : None

Notes : None

Result : Draws debugging axes at the coordinates defined by **x**, **y** and **z**. Each of the 3 possible directions has a different colour :

Direction	Colour
X	Red
Y	Green
Z	Blue

Example :

```
// This example displays the debugging 3D axis arrows at position 0, 0, A10
X_DRAWAXES 0.0, 0.0
```

X_DRAWOBJ

Type : Draw a 3D object

Syntax : `X_DRAWOBJ <index%>, <frame>`

See Also : `X_DRAWANIM`

Notes : None

Result : Draws the 3D object defined by **index%**. If the object is animated, the animation frame can be set by **frame**.

Example :

```
// This example draws object 0, frame 0 at the previously set coordinates
X_DRAWOBJ 0, 0
```

X_ENABLE_Z

Type : Enable or disable Z buffer use

Syntax : X_ENABLE_Z <use%>

See Also : X_CLEAR_Z

Notes :

- X_MAKE_3D re-enables Z buffer use on each call

Result : This command enables or disables the use of the 3D system's Z buffer (depth buffer). If **use% = TRUE** then the Z buffer is used. If **use% = FALSE** then the Z buffer is not used.

Example :

```
// This example disables z buffering
X_ENABLE_Z FALSE
```

X_FOG

Type : Activate or deactivate fogging

Syntax : `X_FOG <colour%>, <mode>, <value1>, <value2>`

See Also : None

Notes : None

Result : Activates or deactivates fogging. Fog is deactivated if **colour%** = -1.

mode defines how fogging is performed, and can be one of the following values :

Value of **mode**	Fog Type	Value of **value1**	Value of **value2**
TRUE	Exponential fog	Density of fog between 0 and 1	Not used
FALSE	Linear fog	ZNear - beginning of fog from camera	Zfar - end of fog

Example :

```
// This example activates grey exponential fog with a density of 0.5
X_FOG RGB(128, 128, 128), TRUE, 0.5, 0
```

X_GETCAMERAMATRIX

Type : Get camera matrix

Syntax : X_GETCAMERAMATRIX matrix[]

See Also : X_CAMERA

Notes : None

Result : This command returns the 3D matrix values used by the last called X_CAMERA command, in the array defined by **matrix**.

The values held in **matrix[]** contain the following value types :

Index in **matrix[]**	Description of value :
0, 1, 2	Direction and scaling of the local X axis of the object to be drawn
3	"W" parameter of X Axis. For normal transformation it is always 0 for the axis, and 1 for the translation.
4, 5, 6	Direction and scaling of the local Y axis of the object to be drawn
7	"W" parameter of Y Axis. For normal transformation it is always 0 for the axis, and 1 for the translation.
8, 9, 10	Direction and scaling of the local Z axis of the object to be drawn
11	"W" parameter of Z Axis. For normal transformation it is always 0 for the axis, and 1 for the translation.
12, 13, 14	Specify the current translation (or displacement), and are a value for the global X,Y,Z position of the object

Example :

```
// Return the matrix value for the last used
X_GETCAMERAMATRIX array[]
```

X_GETCOLLISIONFACE

Type : Get index of a surface that triggered a collision

Syntax : `result% = X_GETCOLLISIONFACE()`

See Also : X_GETFACE

Notes : None

Result : Returns the index of a surface that triggered a collision with the last X_COLLISION*xxxx* command. Face information can be read using X_GETFACE

Example :

```
// This command returns the id of the surface that caused a collision
ID% = X_GETCOLLISIONFACE()
```

X_GETFACE

Type : Retrieve information about a triangle

Syntax : X_GETFACE <index%>, <frame%>, <triIndex%>, <store[]>

See Also : X_NUMFACES

Notes : None

Result : Retrieves the face information about the triangle from the 3D object defined by **index%**.

frame% denotes the frame animation to be used in the 3D object.

triIndex% denotes the triangle index to be used. This value should be between 0 and X_NUMFACES() - 1

The resulting vertices are stored in **store[]**. This creates a 3 x 9 dimensional array, with the first array index holding the index of the node, whilst the second array dimension holds the data in the following way :

face[i][0] = x - X Position
face[i][1] = y - Y Position
face[i][2] = z - Z Position
face[i][3] = tx - X Texture coordinate
face[i][4] = ty - Y Texture coordinate
face[i][5] = color - Colour
face[i][6] = nx - X Normal Vector direction
face[i][7] = ny - Y Normal Vector direction
face[i][8] = nz - Z Normal Vector direction

Example :

```
// The following example does a full analysis of a 3D object :
LOCAL face[]
X_LOADOBJ "wumbo.ddd", 0
X_MAKE3D 1,500,45
X_CAMERA 10,2,-1, 0,2,0
X_DRAWOBJ 0, 2
count = X_NUMFACES(0)
FOR i=0 TO count-1
X_GETFACE 0, 2, i, face[]
// face[node 0..2][x,y,z,tx,ty,col]
X_LINE face[0][0], face[0][1], face[0][2], face[1][0], face[1][1], face[1][2], 4, RGB(0,0,255)
X_LINE face[2][0], face[2][1], face[2][2], face[1][0], face[1][1], face[1][2], 4, RGB(0,0,255)
X_LINE face[0][0], face[0][1], face[0][2], face[2][0], face[2][1], face[2][2], 4, RGB(0,0,255)
X_DOT face[0][0], face[0][1], face[0][2], 8, RGB(0,255,0)
X_DOT face[1][0], face[1][1], face[1][2], 8, RGB(0,255,0)
X_DOT face[2][0], face[2][1], face[2][2], 8, RGB(0,255,0)
NEXT
SHOWSCREEN
MOUSEWAIT
```

X_GETMATRIX

Type : Get transformation matrix

Syntax : `X_GETMATRIX <mat[]>`

See Also : `X_MULTMATRIX`

Notes : None

Result : This command copies the current transformation matrix, that would be used if you called `X_DRAWOBJ`, into the one dimensional array called **mat[].**

The values held in **mat[]** contain the following value types :

Index in **mat[]**	Description of value :
0, 1, 2	Direction and scaling of the local X axis of the object to be drawn
3	"W" parameter of X Axis. For normal transformation it is always 0 for the axis, and 1 for the translation.
4, 5, 6	Direction and scaling of the local Y axis of the object to be drawn
7	"W" parameter of Y Axis. For normal transformation it is always 0 for the axis, and 1 for the translation.
8, 9, 10	Direction and scaling of the local Z axis of the object to be drawn
11	"W" parameter of Z Axis. For normal transformation it is always 0 for the axis, and 1 for the translation.
12, 13, 14	Specify the current translation (or displacement), and are a value for the global X,Y,Z position of the object

Example :

```
// This gets the matrix details for current transformation matrix
X_GETMATRIX array[]
```

X_LINE

Type : Draw a 3D line

Syntax : `X_LINE <x>, <y>, <z>, <dx>, <dy>, <dz>, <width>, <colour%>`

See Also : `X_DOT`

Notes : None

Result : Draws a 3D line from the positions defined by **x**, **y** and **z** to the positions defined by **dx**, **dy**, **dz**.

The width of the line is defined by **width** and colour is defined by **colour%**.

Example :

```
// This example draws a red 3D line from 0, 0, 0 to 100, 110, 50 and with a width of 2 units
X_LINE 0.0, 0.0, 0.0, 100.0, 110.0, 50.0, 2.0, RGB(255, 0, 0)
```

X_LOADOBJ

Type : Loads a 3D object into memory

Syntax : `X_LOADOBJ <fileName$>, <index%>`

See Also : `X_SAVEOBJ`

Notes :

- Only files with extension ".ddd" or ".dda" can be loaded
- Giving an empty filename (or a filename that does not exist) will delete the object from memory

Result : Loads the 3D file called fileName$, into the 3D index **index%**. This value can then be used to display objects.

Example :

```
// This example loads the 3D object into index 1
X_LOADOBJ "PLAYER.DDD", 1
```

X_LOADSHADER

Type : Loads a GLSL shader

Syntax : `result = X_LOADSHADER(<index%>, <vertext$>, <fragment$>)`

See Also : `X_PUTSHADER, X_SETSHADER`

Notes :

- Giving an empty file-name (or a file-name that does not exist) will delete the graphic from memory

Result : Loads a shader file into the index defined by **index%**.

vertext$** defines the file-name for the vertex data, and **fragment$ defines the file-name for the fragment file.

This command returns **TRUE** if the load succeeded or **FALSE** if it didn't.

Example :

```
// This example loads the shader parts "vert" and "frag" into shader index 1
A = X_LOADSHADER(1,"vert","frag")
```

X_MAKE2D

Type : Switch to 2D mode

Syntax : `X_MAKE2D`

See Also : `X_MAKE3D`

Notes : None

Result : Switch to 2D mode so that normal 2D commands can be used

Example :

```
// This example switch back to 2D mode
X_MAKE2D
```

X_MAKE3D

Type : Switch to 3D mode

Syntax : X_MAKE3D <near>, <far>, <fov>

See Also : X_MAKE2D

Notes : None

Result : Switches to 3D mode.

near and **far** defined the distance that objects will be displayed.

fov defines the field of view angle. If this is negative, then an orthogonal view will be created, in which case **fov** will define a scaling value.

Example :

```
// This example activates 3D mode with a near value of 0 units, a far value of 1000 units and a 90
// degree field of view
X_MAKE3D 0.0, 1000.0, 90.0
```

X_MIPMAPPING

Type : Enable or disable mipmapping

Syntax : X_MIPMAPPING <mode%>

See Also : None

Notes : None

Result : Enables (**mode%** = **TRUE**) mipmapping, while it can be disabled by setting mode to **FALSE**

Example :

```
// This example activates mipmapping
X_MIPMAPPING TRUE
```

X_MOVEMENT

Type : Set the position of the next 3D object

Syntax : `X_MOVEMENT <x>, <y>, <z>`

See Also : `X_ROTATION`

Notes :

- This command must be called before `X_ROTATION`, since it will reset all previous rotations

Result : Sets the current drawing position of the next 3D object to the values set by **x**, **y** and **z**.

Example :

```
// This example sets the position for the next 3D object
X_MOVEMENT 0.0, 0.0, 0.0
```

X_MULTIMATRIX

Type : Easy way of using the OpenGL "*glMultiMatrix*" function

Syntax : `X_MULTIMATRIX <store[]>`

See Also : None

Notes :

- **store** must be defined using `DIM` **store[16]**

Result : Calls the OpenGL "glMultiMatrix" function and stores the matrix in the array defined by **store[]**

Example :

```
// This example stores the "glMultiMatrix" values into the array called "ARRAY"
X_MULTIMATRIX ARRAY[]
```

X_NUMFACES

Type : Get number of triangles in a 3D object

Syntax : `result = X_NUMFACES(<index%>)`

See Also : None

Notes : None

Result : Returns the number of triangles in the 3D object defined by **index%**.

Example :

```
// This example retrieves the number of triangles form the 3D object index 0
A = X_NUMFACES(0)
```

X_OBJADDVERTEX

Type : Add a vertex to a 3D object being created

Syntax : X_OBJADDVERTEX <x>, <y>, <z>, <tx>, <ty>, <colour%>

See Also : X_OBJNEWGROUP, X_OBJEND

Notes :

- Triangles n, n-1 and n-2 will be connected. For each group, n-2 triangles will be drawn

Result : This command adds a vertex to the 3D object currently being created.

x, **y** and **z** define vertex coordinates

tx and **ty** set the texture coordinate for this point. 0 is the left/top coordinate and 1 is the bottom/right coordinate

colour% is the colour of this vertex

Example :

```
// This example creates a user-defined 3D object at index 5.  The vertex positions are at -5, -10,
// -5 and the texture positions start at 0, 0.  The colour of this vertex is blue
X_OBJSTART 5
    X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
X_OBJEND
```

X_OBJEND

Type : End creation of a 3D object at runtime

Syntax : X_OBJEND

See Also : X_OBJSTART, X_OBJADDVERTEX, X_OBJNEWGROUP

Notes : None

Result : End the creation of a 3D object.

Example :

```
// This example creates a user-defined 3D object at index 5.  The vertex positions are at -5, -10,
// -5 and the texture positions start at 0, 0.
// The colour of this vertex is blue
X_OBJSTART 5
   X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
X_OBJEND
```

X_OBJNEWGROUP

Type : Create a new group of vertices

Syntax : X_OBJNEWGROUP

See Also : X_OBJSTART, X_OBJADDVERTEX, X_OBJSETFRAME, X_OBJEND

Notes :

- This commands must be used between X_OBJSTART and X_OBJEND

Result : Finish the current vertex group ready for a new one if needed

Example :

```
// This example creates a user-defined 3D object at index 5.  It has two vertices - one at at -5,
// -10, -5, with texture coordinates at 0,0 and a colour of blue.
// The other vertex is also at -5, -10, -5, with texture coordinates at 0,0 and a colour of blue
X_OBJSTART 5
    X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
    X_OBJNEWGROUP
    X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
X_OBJEND
```

X_OBJSETFRAME

Type : Switch to another animation frame

Syntax : X_OBJSETFRAME id%

See Also : X_OBJSTART, X_OBJADDVERTEX, X_OBJEND

Notes :

- This commands must be used between X_OBJSTART and X_OBJEND

Result : Switch to a new animation frame, defined by **id%**, allowing the creation of animated 3D objects pro grammatically

Example :

```
// This example creates a user-defined 3D object at index 5.  It has two vertices - one at at -5,
// -10, -5, with texture coordinates at 0,0 and a colour of blue.
// The other vertex is also at -5, -10, -5, with texture coordinates at 0,0 and a colour of blue
X_OBJSTART 5
    X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
    X_OBJNEWGROUP
    X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
X_OBJEND
```

X_OBJSTART

Type : Create a 3D object at runtime

Syntax : X_OBJSTART <index%>

See Also : X_OBJADDVERTEX, X_OBJNEWGROUP, X_OBJSETFRAME, X_OBJEND

Notes : None

Result : Start the creation of a 3D object.

index% defines the ID number of the object to be created.

Example :

```
// This example creates a user-defined 3D object at index 5.  The vertex positions are at -5, -10,
// -5, nd the texture positions start at 0, 0.   The colour of this vertex is blue
X_OBJSTART 5
   X_OBJADDVERTEX -5, -10, -5, 0,0,RGB(0,0,255)
X_OBJEND
```

X_POPMATRIX

Type : De-group objects

Syntax : X_POPMATRIX

See Also : X_PUSHMATRIX

Notes : None

Result : De-group a set of objects allowing each object to be positioned independently of each other

Example :

```
// This example groups a position command (to 0, 0, 0), a scaling command (to 0.3, 0.3, 0.3) and a
// drawing command are grouped together and relative to
// the previously set position commands
X_PUSHMATRIX
X_MOVEMENT 0.0, 0.0, 0.0
X_SCALING .3, .3, .3
X_DRAWOBJ 0, 0
X_POPMATRIX
```

X_PRINT

Type : Draws 3D text

Syntax : X_PRINT <text$>, <x>, <y>, <z>, <scale>

See Also : None

Notes : None

Result : Draws the contents of text$ at the position defined by **x**, **y** and **z**. These coordinates are the bottom centre of the text.

scale defines the scaling value. If this is 0.0, then the original text height will be used and the text will not be smaller the further away it is.

Example :

```
// This example displays the text  "THIS IS A TEST" at position 0,0,0 and scaled by a factor of 1.2
X_PRINT "THIS IS A TEST", 0.0, 0.0, 0,0, 1.2
```

X_PUSHMATRIX

Type : Group objects

Syntax : X_PUSHMATRIX

See Also : X_POPMATRIX, X_MOVEMENT, X_SCALING, X_ROTATION

Notes : None

Result : Group a set of objects. All X_MOVEMENT, X_SCALING and X_ROTATION commands are relative to the previously set position commands. This allows the placing of further objects until X_POPMATRIX is used whereby all objects are drawn as one unit.

Example :

```
// This example groups a position command (to 0, 0, 0), a scaling command (to0.3, 0.3, 0.3) and a
// drawing command are grouped together and/ relative to the previously set position commands
X_PUSHMATRIX
X_MOVEMENT 0.0, 0.0, 0.0
X_SCALING .3, .3, .3
X_DRAWOBJ 0, 0
X_POPMATRIX
```

X_PUTSHADER

Type : Allows changing of the uniform float value in a shader

Syntax : X_PUTSHADER <name$>, <value>

See Also : X_LOADSHADER, X_SETSHADER

Notes : None

Result : This command allows the "uniform float" variable in the current shader program to be set to the value defined by **value**.

name$ is the shader name.

Example :

```
// This sets the uniform float variable inside the shader.shd program to 1
X_PUTSHADER "shader.shd", 1
```

X_ROTATION

Type : Sets the rotation angle for the next 3D object

Syntax : X_ROTATION <angle>, <x>, <y>, <z>

See Also : X_MOVEMENT

Notes : None

Result : Rotates the next 3D around the axis defined by **x**, **y** and **z**, by the angle defined by **angle**. Several rotations can be applied to an object

Example :

```
// This example rotates the next drawn object by 32 degrees in the Z direction
X_ROTATION 32.0, 0, 0, 1.0
```

X_SAVEOBJ

Type : Save a 3D object to a file

Syntax : X_SAVEOBJ <fileName$>, <index%>

See Also : X_LOADOBJ

Notes : None

Result : Save the 3D object defined by **index%** to the file defined by **fileName$**

Example :

```
// This examples save the 3D object from index  1 to the file "3D.DDD"
X_SAVEOBJ "3D.DDD", 1
```

X_SCALEMODEL

Type : Scale a model

Syntax : X_SCALEMODEL <obj%>, <scx>, <scy>, <scz>

See Also : X_SCALE

Notes :

- This command may take some time to complete

Result : Re-scales the 3D object, held in **obj%**, by the values held in **scx**, **scy** and **scz**, which correspond to the X, Y and Z directions of a model. This allows collision routines to work properly with a scaled object.

Example :

```
// This example scales model 1 to 2 twice its size
X_SCALEMODEL 1,2.0,2.0,2.0
```

X_SCALING

Type : Sets the scaling size for the next 3D object

Syntax : X_SCALING <x>, <y>, <z>

See Also : X_MOVEMENT, X_ROTATION

Notes : None

Result : Sets the scaling values for the next 3D object.

x, **y** and **z** define the x, y and z scaling values.

Example :

```
// This example scales the next object to be draw by 1.6, 1.2 and 1.0
X_SCALING 1.6, 1.2, 1.0
```

X_SCREEN2WORLD

Type : Maps 2D coordinates to 3D coordinates

Syntax : X_SCREEN2WORLD <sx>, <sy>, <sz>, <wx>, <wy>, <wz>

See Also : X_WORLD2SCREEN

Notes : None

Result : Maps the 2D coordinates defined by **sx**, **sy** and **sz** into 3D world coordinates stored in **wx**, **wy** and **wz**

Example :

```
// This example converts the 2D coordinates 100, 100 to 3D ones and stores the result in WX, WY and
// WZ
X_SCREEN2WORLD 100.0, 100.0, 0.0, WX, WY, WZ
```

X_SETCELSHADES

Type : Define level of brightness for a shader

Syntax : X_SETCELSHADES <shade[]>

See Also : X_LOADSHADER, X_SETSHADER, X_PUTSHADER

Notes :

- This function should be called at the beginning of the program only once as all textures need to be recreated when they are next rendered. As a result of the texture recreation you will experience a small delay for every call to the function

Result : Set steps of brightness to be used for cell shading. The levels are stored in the shader array **array[].** The values in **array[]** should be between 0.0 and 1.0 and define the brightness at each reflection angle.

Example :

```
// This example passes the brightness values held in ARRAY to the next object that uses cell shading
X_SETCELSHADES ARRAY[]
```

X_SETSHADER

Type : Sets the shader to be used

Syntax : X_SETSHADER <index%>

See Also : X_LOADSHADER, X_SETCELSHADES, X_PUTSHADER

Notes : None

Result : Use the shader defined by **index%**.

Example :

```
// This example sets shader 0 to be used
X_SETSHADER 0
```

X_SETTEXTURE

Type : Set texture to be used on a 3D object

Syntax : X_SETTEXTURE <index%>, <bump%>

See Also : LOADBUMPTEXTURE

Notes :

- A bump map texture should be loaded using LOADBUMPTEXTURE

Result : Use the sprite ID defined by index to be used on the next 3D object. If **index%** = -1 then no texture will be used.

bump% specifies which normal map will be used for dot3 bump mapping. If **bump%** = -1 then no bump mapping is used.

Example :

```
// This examples uses sprite 0 as a texture, and sprite 1 and a bump-map image
X_SETTEXTURE 0, 1
```

X_SETTEXTUREOFFSET

Type : Shift texture polygons

Syntax : X_SETTEXTUREOFFSET <x>, <y>

See Also : None

Notes : None

Result : Shifts the texture polygons by the values defined by **x** and **y**.

Example :

```
// This sets the texture offset to 0, A10
X_SETTEXTUREOFFSET 0.0, 0.0
```

X_SPHEREMAPPING

Type : Enable or disable sphere mapping

Syntax : X_SPHEREMAPPING <status%>

See Also : None

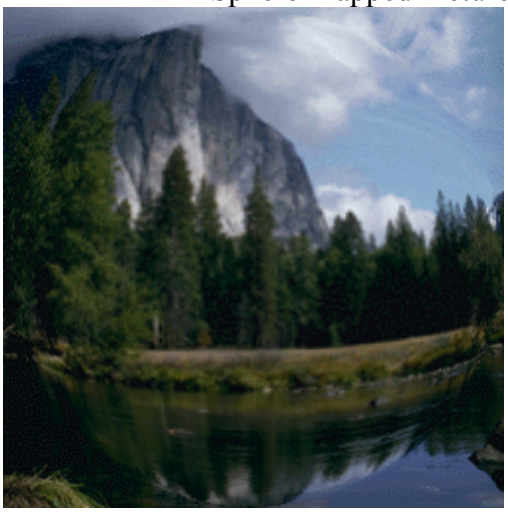
Notes :

- In order to properly use this command, you must load a spherical projection of the environment to a sprite index

Result : If **status%** = **TRUE**, then sphere mapping is enabled. This allows you to create metallic reflections
If **status%** = **FALSE** then no sphere mapping is performed.

Original Picture

Sphere Mapped Picture



Example :

```
// This example activates sphere mapping
X_SPHEREMAPPING TRUE
```

X_SPOT_LT

Type : Enable spot lighting

Syntax : `X_SPOT_LT <index%>, <colour%>, <x>, <y>, <z>, <dirX>, <dirY>, <dirZ>, <cutoff>`

See Also : `X_CULLMODE`

Notes :

- When using special effects `X_CULLMODE` should be set to either -1 or 1

Result : Create a spot light, which is defined by **index%**, which should be a value between 0 and 7 for normal lights and -3 to -1 for special effects :

Value of **index%**	Result
-1	**Bump Mapping** A second texture will be used as a bump map texture. Darker pixels on the texture represent a lower point, and brighter pixels represent a higher one. Only **x**,**y** and **z** light position coordinates are taken into account. You must load textures with `LOADSPRITE` and `LOADBUMPTEXTURE` and select them with `X_SETTEXTURE` before use.
-2	**Cell or Toon shading (aka Cartoon Rendering)** With this option, a light calculation will be performed that does not result in gradual light changes, but in stepped levels of brightness, resembling a cartoon. A texture set by `X_SETTEXTURE` will be overlayed on the object. This can be used to model faces and details onto a character. Only the **x**, **y**, and **z** light position coordinates are taken into account.

-3	**Shadows** This creates real-time shadows using the stencil buffer. The whole scene should be rendered first, then this command used, followed by the rendering of all objects again. **colour%** is used to taint the shadows

colour% defines the colour of the light

x, **y** and **z** defines the position of the light

dirX, **dirY** and **dirZ** specify the direction vector that the light shines along

cutoff specifies the light's cut-off angle. This should be a value of :

Value of **cutoff**	Result
0	Turn light off
1 – 179	Spot light
180	Parallel light source
360	Point light source

Example :

```
// This example creates a spot light with an index of 0.  It is a dark yellow colour and projects
// from 0, 0, 0 in the direction of 1, 8, 0.  It is a point light
X_SPOT_LT 0, RGB(255,0,128), 0.0, 0.0, 0.0, 1.0, 8.0,0.0, 360
```

X_SPRITE

Type : Draws a 2D sprite at a 3D position

Syntax : X_SPRITE <sprite%>, <x>, <y>, <z>, <scale>

See Also : None

Notes : None

Result : Draws a 2D sprite at the 3D position defined by **x**, **y** and **z**. **scale** defines the scaling value of the sprite, and **sprite%** defines the graphic to be used.

Example :

```
// This example draws sprite index 0 at coordinates 0, 0, 0 and scales it by a factor of 1.2
X_SPRITE 0, 0.0, 0.0, 0.0, 1.2
```

X_WORLD2SCREEN

Type : Converts a 3D position to 2D screen coordinates

Syntax : X_WORLD2SCREEN <wx>, <wy>, <wz>, <sx>, <sy>, <sz>

See Also : X_SCREEN2WORLD

Notes : None

Result : Converts the 3D coordinates defined by **wx**, **wy** and **wz** into 2D coordinates, which will be stored in **sx**, **sy** and **sz**

Example :

```
// This example converts the 3D coordinates 100, 100, 25 to 2D values and stores them in SX, SY
// and SZ
X_WORLD2SCREEN 100.0, 100.0, 25.0, SX, SY, SZ
```

ZOOMSPRITE

Type : Draw a rescaled sprite
4
Syntax : `ZOOMSPRITE <index%>, <x>, <y>, <scaleX>, <scaleY>`

See Also : None

Notes :

- The coordinates will position the centre of the scaled sprite, unless the scaleX or scaleY values are 1, at which point the left side (scaleX = 1.0) and/or top part (scaleY = 1.0) is used instead

Result : Resizes a sprite from sprite index **index%**. Horizontal positioning is set by **x**, and vertical positioning is defined by **y**.

The sprite with be drawn with the new horizontal scale value of **scaleX** and the new vertical scale value of **scaleY**.

Example :

```
// This rescales sprite 1 by 1.5 and 1.5 and positions it at coordinates at 100 by 100
ZOOMPRITE 1, 100, 100, 1.5, 1.5
```

bAND

Type : Returns result of a binary AND operation

Syntax : `result = bAND(<value1%>, <value2%>)`

See Also : bOR, bNOT, bXOR

Notes : None

Result : Returns the value of a binary AND operation of **value1%** and **value2%**

Example :

```
// This sets A to be 0
A = bAND(16, 4)
```

bNOT

Type : Returns result of a binary NOT operation

Syntax : `result = bNOT(<value%>)`

See Also : bAND, bOR, bXOR

Notes : None

Result : Returns the value of a binary NOT operation of **value%**

Example :

```
// This sets A to be -17
A = bNOT(16)
```

bOR

Type : Returns result of a binary OR operation

Syntax : `result = bOR(<value1%>, <value2%>)`

See Also : bAND, bNOT, bXOR

Notes : None

Result : Returns the value of a binary OR operation of **value1%** and **value2%**, into result

Example :

```
// This sets A to be 36
A = BOR(32, 4)
```

bXOR

Type : Returns result of a binary exclusive-OR operation

Syntax : `result = bXOR(<value1%>, <value2%>)`

See Also : bAND, bOR, bNOT

Notes : None

Result : Returns the value of a binary XOR operation of value1 and value2

Example :

```
// This sets A to be 40
A = BXOR(32, 8)
```

get_sprite_texture

Type : Get OpenGL texture ID

Syntax : `result% = get_sprite_texture(<sprite%>)`

See Also : `INLINE`

Notes :

- This command is usually used with `INLINE`

Result : Returns the id number of the sprite defined by **sprite%**

Example :

```
// Get ID of sprite texture 1
INLINE
int id=get_sprite_texture(1)
ENDINLINE
```

self

Type : Own type operator

Syntax : `self.variable`

See Also : TYPE, ENDTYPE, "`.`"

Notes :

- Can only be used with functions that are in a TYPE

Result : This command allows a function in a type to access any variables that have been defined in the TYPE. Without this, the compiler would try to use a global variable. It also allows a function in a type to call another function in a type - failing to do so usually results in a compiler error.
The variable to be access is defined by variable

Example :

```
// THIS EXAMPLE ACCESSES THE VARIABLE "EXAMPLE%" THAT IS IN THE TYPE
TYPE TExample
   example%

   FUNCTION Initialise%:
       self.example%=1 // Use example% held in TExample
   ENDFUNCTION

   FUNCTION test%:
       example% // This will access the global "example" variable
   ENDFUNCTION
ENDTYPE
```

Access member of a type (.)

Type : Type operator

Syntax : `type.variable`

See Also : TYPE, ENDTYPE

Notes :

- Don't confuse the full-stop with a comma (,) , otherwise you will get a compiler error

Result : In GLBasic, the only way to access a member of a TYPE, whether its a single one or in an array

Example :

```
// To access a variable in a single type
DEBUG value.one%

// To access a variable in an array
DEBUG value[5].one%
```

Comment (//)

Type : Comment text

Syntax : *// [string]*

See Also : N/A

Notes :

- When commenting multiple lines, each one has to have // on it

Result : This operator is used to denote that any characters after the // characters are to be ignored – this also include GLBasic keywords.

Example :

```
// This line is ignored
// PRINT "A",1,1,FALSE - This line is also ignored
```

Index

Section.....Page

Introduction.....3
Tutorial #1.....5
Tutorial #2.....15
Tutorial #3.....20
Tutorial #4.....22
Tutorial #5.....23
Tutorial #6.....26
Tutorial #7A.....30
Tutorial #7B.....34
Tutorial #8.....39
Tutorial #9.....47
Tutorial #10.....48
Tutorial #11.....50
Anatomy of a GLBasic Program.....58
Executable sizes.....59
2D Graphics.....62
Text Font.....63
Sprites.....64
3D Graphics.....65
Graphics In General.....67
Sounds And Music.....68
File I/O.....69
Using C/C++ (or perhaps another language).....70
Optional Subroutines.....79
Variables.....81
Compiling.....83
Compiling for webOS.....84
Compiling for Android.....98
Movement And Animations.....103
Fading Up and Down.....104
Portrait and Landscape.....105
Icons And Graphics.....106
Network Lobby.....107
Limitations.....109
Programming Guidelines.....113
Style Guidelines.....115
Installation Problems.....116
Error Messages.....117
Available Commands in GLBasic.....122
Available Commands in GLBasic (Category Order).....135
- ?.....139
- A.....145
- B.....156
- C.....161
- D.....170
- E.....186
- F.....193
- G.....202
- H.....224
- I.....225
- J.....236
- K.....237
- L.....239
- M.....252
- N.....260
- O.....272
- P.....274
- R.....284
- S.....298
- T.....339
- U.....344
- V.....348

Index (Cont)

Section.....Page

W.....349
X.....355
Z.....390
b.....391
g.....393
s.....394
Member Operator(.).....395
Comments (//).....396